Electronic version of an article published as **Requirements Engineering Journal, Vol. 13, No. 1, Jan. 2008, pp. 73-86**

[doi: 10.1007/s00766-007-0058-9]

MOQARE: Misuse-oriented Quality Requirements Engineering

Abstract. This work presents MOQARE (Misuse-oriented Quality Requirements Engineering), a method to explore quality requirements. The aim of MOQARE is to support intuitive and systematic identification of quality requirements. It was developed by integrating and adapting concepts from other methods (like Misuse Cases). It provides a general conceptual model of quality requirements, and a checklist-based process for deriving them in a top-down fashion. This derivation starts from business goals and vague quality requirements and delivers detailed requirements. MOQARE applies to requirements on the system to be developed requirements, but also derives requirements on the development process, including administration and maintenance. It considers normal and extreme use. The relationships among these requirements are modeled in a Misuse Tree. MOQARE has shown its merits in several case studies, one of which is presented here.

Keywords: requirements elicitation, requirements specification, non-functional requirements, Misuse Cases, quality requirements

1 Introduction

The elicitation of quality requirements (QR) for software systems – also called non-functional requirements - often starts with quality goals vaguely expressed which apply to the whole system (e.g.: “The system has to be secure/ easy to use/ fast.”). However, such requirements are neither detailed enough for being implemented by a developer nor specific enough for being verified by a tester, nor are they useful for cost estimation. Therefore, we developed the method MOQARE (Misuse-oriented Quality Requirements Engineering) which derives QRs from business goals and details vague QRs into system specific, realizable and verifiable requirements.

Other methods of exploring QRs often are designed for a specific quality attribute or for a specific activity. E.g. Misuse Cases detail security requirements, and ATAM (Architecture Tradeoff Analysis Method) supports the evaluation of given architectural alternatives with respect to QRs. The NFR Framework is applicable to all types of QRs, but focused on the documentation and negotiation of QRs, and not on their elicitation from business goals. However, MOQARE is applicable to all types of QRs, systematic and thorough, and at the same time readily comprehensible for non-technical stakeholders.

This publication describes MOQARE´s general conceptual model of QRs and the checklist-based process for deriving them in a top-down fashion. This process was evaluated and improved in its application in several case studies. One important case study is presented here.

The article is structured as follows: Section 2 presents the case study setting, since we use examples from the case study when we present our conceptual model in

section 3 and describe the process of requirements elicitation and specification with MOQARE in section 4. The case study results are presented in section 5. Lessons learned are summarized in section 6. Section 7 discusses related work. In section 8, we conclude with perspectives for further research.

2 Presentation of the Case Study Setting

MOQARE in a case study was applied to the "Uveitis Database" used at the Interdisciplinary Uveitis Center Heidelberg. This Center was founded in 2001 at the Otto-Meyerhof-Zentrum. Ophthalmologists and internists work together to diagnose and treat the non-trivial causes of uveitis, an inflammatory eye disease. The Uveitis Database is used by doctors, nurses and reception personnel to manage patient data such as address and age as well as examination results, diagnoses, medication and operation data at different points of time, both for the analysis and optimization of the therapy course, and as a data basis for scientific studies. Data are entered manually at client computers, or – tentatively – on a mobile device. The software is in operative use, and is being improved and enhanced constantly. In addition to the Uveitis Database, the documentation of the patients and their data is also being filed in paper.

MOQARE in this case study was used to analyse the QRs of the Uveitis Database. Knowing the business goals to be supported by the Uveitis Database, we identified those software quality attributes (QA) which serve these business goals best, e.g. integrity and privacy of data. From these, countermeasures were derived. Countermeasures are the detailed requirements and the intended result of the MOQARE analysis, as they allow to define precisely what quality means in the case of the Uveitis Database and they allow to improve the quality of this software.

3 Conceptual Model of MOQARE

Quality requirements are difficult to elicit [1]. Misuse Cases have proven to be intuitive. Therefore, we chose the Misuse Case approach as a basis for detailing QRs from business goals down to quality goals and further to detailed requirements (here called 'countermeasures'). The original Misuse Case principle is: Misuse Cases foresee the behaviour of a misuser whose activity (or inactivity) constitutes or leads to misuse of the system or even whose goal is to misuse the system. Such misuse cases can be defined as "a function that the system should not allow" [2] or as the description of threats to (security) goals [3]. From these Misuse Cases new requirements can be derived which prevent or mitigate misuse. In this way, Misuse Cases help to complement the system specification, also considering extreme use, exceptions and threats in addition to intended normal and successful use. The application of Misuse Cases in the area of security has demonstrated that these considerations help the stakeholders to define in detail what security means in their domain and how this desired security level can be achieved by technical and organizational requirements.

MOQARE [4] identifies potential Misuse Cases not only with respect to security, but with respect to all QAs and derive further requirements thereof. For instance, one threat to data integrity, is unintentional data corruption by users, and therefore those software properties and other measures which prevent user errors support data integrity. In the application of Misuse Cases to all QAs, we found that a more general terminology is necessary, because Misuse Cases are tailored to security requirements. As will be shown in section 7, Misuse Cases and several other methods of exploring QRs, are based on the same general principle: An asset is to be protected from a threat, and to do so, countermeasures are defined. Therefore we think it is appropriate to adopt concepts from these other methods. Figure 1 presents an overview of the MOQARE concepts and their relationships. Table 1 lists examples for most of the concepts as well as categories, on the highest level of granularity.

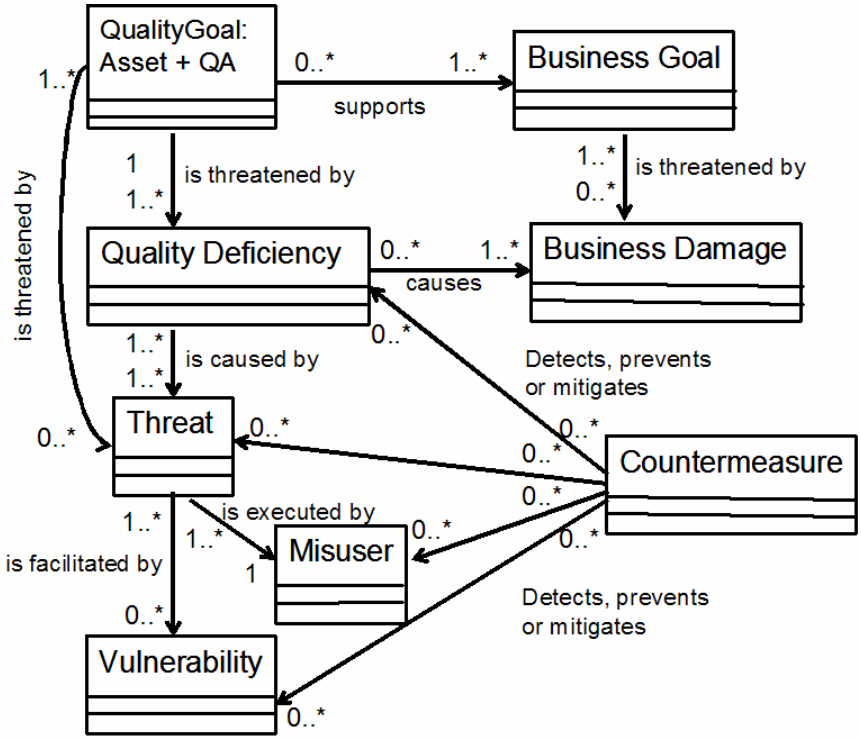

Figure 1: The MOQARE concepts: see text for definitions and explanation

A system is developed and used because it supports important *business goals*. Therefore, these business goals justify all other requirements on the system. The business goals usually are threatened by *business damages* which are partly caused by *quality deficiencies* of the system.

The business goals are supported by *quality goals* of the system. A quality goal is the combination of an asset plus a QA, which both are to be protected, like “integrity of the data”. (Remark: A QA describes an aspect or characteristic of quality, while QRs are instantiations of QAs, i.e. requirements which refer to a defined system.) An *asset* can be any part of the system. The quality goals are high-level QRs. Although stakeholders (e.g. users) usually cannot tell how the system can support their business goals, they can define the relevant QA.

A *quality deficiency* means that the asset does not satisfy the QA. The quality deficiencies concretize how (when/ where/ how much) the system does not satisfy the QA. This non-compliance can be total or partial, permanent or temporary. For example, if the quality goal is “availability of data”, the quality deficiency can consist in temporary inaccessibility for all users or for certain users, irreversible destruction of the data, corruption of the data, and many more.

A *threat* is an action (during system use, development, administration or maintenance) which causes a quality deficiency and consequently degrades the satisfaction of a quality goal. For instance, testing which neglects usability testing threatens the usability of the user interface. The threat is usually executed by a *misuser*, its driving force. In literature, the misuser often is described in terms of a misuser motivation, goal or attribute. This might be business damage (i.e. pure destruction), an advantage for himself like enrichment, or a characteristic of the misuser (being disgruntled or careless). As we do not want to mix goals, damages, etc., we either add this information to the misuser description, like in “disgruntled employee” or in the Misuse Case (as precondition). To identify misusers and threats, not only normal uses (typical uses of the existing system) are relevant, but also growth scenarios (these cover anticipated changes of the system; relevant for maintainability, interoperability, safety and portability) and exploratory scenarios (these cover extreme changes; relevant for security, reliability, efficiency, recoverability). These three types of scenarios were proposed by ATAM [5].

Often, the threat is facilitated or even provoked by a *vulnerability*. A vulnerability is a property of the system, either a flaw or a side-effect of an otherwise wanted property, if it can be misused with respect to a quality goal. For instance, concerning the quality goal “data integrity”, a possible vulnerability can be “lack of usability of user interface where users enter data”. The corresponding threat then is the unintentional data corruption (by user error) and the consequence is wrong data. On the other hand, not each flaw is a vulnerability. If there is no potential misuser for it, then the flaw is not a vulnerability. So, each system flaw and system property must be evaluated regarding the quality goal, to decide whether there is a potential misuser who might threaten the QA of the asset.

How, why and by whom a quality deficiency is caused and quality goal threatened, is documented in the form and granularity of a Misuse Case which is similar to a Use Case. Misuse Cases can be modeled as separate Misuse Cases, but also as an exception scenario as part of a Use Case. In the Misuse Case, the misuser corresponds to the Use Case´s actor, the vulnerability is a pre-condition, the steps of the threat are described by a scenario, and the consequences (quality deficiency) are a post-condition of the Misuse Case. If we call the vulnerability a pre-condition, we assume that the vulnerability is a necessary (but not sufficient) condition for the threat. There might also be vulnerabilities which are not a pre-condition but aggravate a misuse.

An example of a Misuse Case, which refers to the quality goal “data integrity”, is the following:

misuser: mobile device

pre-condition: configuration of data synchronization set on “data set creation” (vulnerability); user has entered data on mobile device and another user has entered data concerning the same patient on Uveitis Database

threat: creation of doublets during synchronization of Uveitis Database with mobile device

scenario:

- mobile device connects to Uveitis Database and starts synchronization
- data conflict is detected, and it is unclear which data are more recent
- new data set is created in the Uveitis Database, containing the data entered on the mobile device

post-condition: two data sets exist for the same patient

Table 1: Categories and examples of each MOQARE concept

Concept	Categories	Examples
Business goals and business damages	They belong to the following five dimensions [6]: product size, quality, staff, cost, (calendar) time.	good therapy for patients at optimal cost; valid scientific studies
Asset	An asset is any part of the system. By “system” we do not only include the software, hardware, and network, but also the physical building, the company, the personnel of the system, and the development, operation and maintenance process. Firesmith [7] lists: “data, communications, services, hardware components, and personnel”. Architecture analysis methods like ATAM mainly consider architecture components. For us, assets can be all of these.	Individual patient data
QA	functionality, reliability, usability, maintainability, portability, efficiency; they can be detailed further, see the hierarchy of ISO 9126 [8]	Integrity; privacy
Vulnerability	a property of the system (the code, design or software development, operation or administration process), either a flaw or a side-effect of an otherwise wanted property, if it can be misused with respect to a quality goal	lack of usability of user interface where users enter data; time-pressure
Threat	theft, intentional destruction, accident, environment change, error during usage, development or maintenance	Unintentional corruption of data; testing which neglects usability testing

Misuser	A misuser can be a person, other systems or forces of nature like fire and thunderstorm. Security is a special case, where the misuser either is an intruder or a user who executes a Use Case he/ she is not supposed to, following a harmful goal, or a careless user violating security rules. All other QAs are threatened by regular system personnel (e.g. end users, administrators and maintainers, and developers) who try to use the system as intended, but fail for some reason.	User (like doctors, nurses and reception personnel), developer, maintainer
Counter-measure	Countermeasures can be new functional requirements (e.g. Use Cases), new or extended exception scenarios of Use Cases, QR on Use Case (including metrics if possible), architectural constraints, user interface constraints, constraints on project and software development, constraints on administration or maintenance, or another quality goal [5][9]-[12]	

The objective of the MOQARE analysis is to derive realizable requirements which describe how the quality goals can be achieved. These requirements are obtained by identifying *countermeasures* for misuses. Countermeasures can counteract the threat, the misuser or his/her motivation, the vulnerability or that the Misuse Case leads to the predicted negative consequences (e.g., quality deficiency). Countermeasures can either detect, prevent [2] or mitigate [13]. They might reduce or eliminate the risk, i.e. the probabilities or the damage severity. Some countermeasures derived, though, are not realizable, but are on the high detail level of quality goals and therefore demand further analysis. As the countermeasures can be of different types, each countermeasure is described by a parameter "type". Is it a new functional requirement (FR), a quality goal, a QR on an FR, an architectural constraint, etc.? This parameter prepares the integration of the MOQARE concepts with the FRs.

The MOQARE concepts can be collected in a Misuse Tree, i.e. a hierarchical graphical presentation which displays the relationships among the concepts. For an example see Figure 2.

4 The MOQARE Process

In this section, we describe how we elicit the MOQARE concepts described in section 3. In MOQARE, the resulting Misuse Tree is more important than the process itself, which we think must be as flexible as possible, as requirements elicitation is a creative activity.

MOQARE starts with a set of (usually incomplete) FR of a planned or existing system. The requirements engineer is guided by a four-step process and supported by checklists. These checklists represent a compilation of reusable knowledge. It was

gained by classifying the content of checklists collected from literature and is enhanced by our case studies. (As Firesmith [14] discusses, security requirements are potentially highly reusable. In the course of our case study, we saw that this is also true for other QRs.) Table 1 presents the categories for each concept on their highest level of granularity. The checklists and their sources are published in a technical report [15]. Separate checklists help to identify each of the following concepts: business goals, QAs, assets, and misusers. The most extensive lists are these two: a list of threats, misusers and their countermeasures for each QA; and a list of vulnerabilities and their countermeasures for several assets, like data or personnel in general, or specific operating systems or types of software products. More details about content and structure of these checklists are given later in this section.

MOQARE recommends the identification of the concepts in the following order:

1. Find the quality goals (based on business goals, business damages, and quality deficiencies).
2. Describe Misuse Cases (including threat, misuser, vulnerabilities, consequences).
3. Define countermeasures.
4. With countermeasures which are quality goals, re-start the cycle at step 2.

This information can be elicited from different stakeholders but also obtained by document analysis or software (prototype, legacy system) analysis.

These four steps are now described in more detail:

1.) A **quality goal** is valuable because it supports a *business goal*. Therefore, it makes sense to start with the definition of the business goals. This step is guided by the questions: What is essential to this business? Why is this IT system being developed? The five dimensions named in table 1 are useful here. In our case study, the business goals were “good therapy for patients at optimal cost” and “valid scientific studies”.

After having identified the major business goals, the next question is which *business damages* might threaten them. To identify the *quality attribute QA* to be protected, we use the ISO 9126 hierarchy of the QAs as a checklist. We use the two levels of this standard and include a third level for security [15] as is common in the security community. Probably all QAs must be satisfied to a certain degree, but which of them have essential influence on the business goals and business damages? From these QAs, the *quality deficiencies* are derived as well as the *quality goals*.

The *quality goals* are derived from the QAs by adding the affected *assets*. Each QA can apply to several assets, and the same asset can be protectable with respect to several QAs. The asset identification is supported by a high-level checklist (as in table 1) and a more detailed hierarchy of assets. They are far from being exhaustive, but they help to identify the specific assets in a given system.

As can be seen in Figure 1, the relationships between the concepts are complex. For each business goal, the business damages and quality deficiencies can be identified that may threaten it (as has been done above) or, vice versa, it can be considered which quality goals support the business goal. Both ways will lead to the quality goals which describe high-level QRs for the system.

2.) **Describe** Misuse Cases: the misuser is the actor, the vulnerability a pre-condition, the steps of the threat are described by a scenario, and the consequences (quality deficiency) are a post-condition of the Misuse Case.

As the same threat can be performed by several different misusers following different courses of events (e.g. data corruption can be intended by an intruder, but also be caused by a user making an error, or by corruption by a system breakdown, system errors, software engineering errors), we start with the identification of the *threats* and then derive one Misuse Case per misuser. The Misuse Case creation is supported by checklists: for each QA, specific threats are listed and for each of them resulting quality deficiencies, potential misusers and countermeasures. For example, theft threatens availability and confidentiality, but normally not usability or efficiency. One can and should use domain specific roles like "ophthalmologist" to describe the misuser in more detail than in the list.

For a given asset, e.g. a certain operating system, potential *vulnerabilities* are known. Therefore, we also use lists of vulnerabilities and their countermeasures. (Our main source of software, hardware and personnel vulnerabilities and their countermeasures was a handbook of the German Ministry for Security in Information Technology [16]; concerning the software engineering process the works of Nakajo and Kume [17] and of Lutz [18]). They can be used as checklists to consider the most common vulnerabilities and to trigger ideas for further vulnerabilities and Misuse Cases not identified on the basis of the threat lists. We think that the vulnerability lists are relevant to all QAs and depend mainly on the asset, as the same property/ vulnerability can often be misused with respect to different QAs.

3.) **Define countermeasures**: For each Misuse Case, one tries to find countermeasures for the threat, the misuser, the misuser´s motivation, the vulnerability and the caused quality deficiency. Our lists of threats and vulnerabilities also provide countermeasures, but we do not claim them to be complete. Moreover, they are quite general, while the ideal countermeasure is concrete, realizable and system-specific. Therefore, the lists are intended to be used to trigger a system-specific brainstorming. It is important to add a metric to the countermeasure where possible. For instance, countermeasures against user errors are usability and fault tolerance of user interfaces, testing, compliance to standards, plausibility check of user input, training, and many more.

4.) If necessary, **re-start the cycle** at step 2: A countermeasure can also be a new quality goal. For example, the usability of a user interface helps to improve the integrity of the data, if they are entered by the users manually. In this case, the elicitation of QRs is not finished with finding all countermeasures, but has to start a new Misuse Case analysis referring to the newly defined quality goals.

The MOQARE results can be presented in the form of a graph, a "Misuse Tree" (see in Figure 2 an extract from the case study results), similar to attack trees of van Lamsweerde et al. [19] or quality models [10], [20]. They also derive detailed requirements from the QA top-down, but – compared to MOQARE – they lack some of our concepts, as is discussed in section 7.

A Misuse Tree has the following levels, from top to bottom:

- business goal
- business damage
- quality deficiency
- quality goal
- Misuse Case (including threat, misuser, vulnerability, consequences)
- countermeasure, some of them being quality goals
- Misuse Case
- countermeasure, some of them being quality goals
- …

A graphical tree presentation appears to be natural because for each business goal, there are several business damages, quality deficiencies and quality goals, for each quality goal, several misuses and for each misuse several countermeasures. Note that we call it a tree because of the clear levels. However, strictly speaking, it can be a graph, see for example Figure 2, where the same quality deficiency leads to several business damages. Often, the same countermeasure applies to several Misuse Cases.

Two alternative strategies can be used for the MOQARE analysis: Aiming at a complete analysis of all types of potential threats (actual, future, already prevented, improbable), all potential threats are considered, and the probability and the damage are estimated later-on. However, in practical work, the efficient analysis of the actual quality of the system (or system prototype or design) and the potential ways for effective improvements are often more desirable. In this case, threats or vulnerabilities need not be considered where an effective countermeasure is already intended, or if the threat/ vulnerability is unlikely or not dangerous for other reasons. In the case study, we concentrated on relevant misuses, i.e., those with significant probability and those which may cause the most harmful damage.

5 Case Study

As mentioned in section 2, MOQARE was applied to the Uveitis Database which is in operation at the Interdisciplinary Uveitis Center Heidelberg. MOQARE was used to describe the QRs of the actual system and to elicit requirements which improve the system quality.

The analysis of the requirements of the Uveitis database was performed by two persons: 1.) the MOQARE specialist and 2.) the requirements engineer, developer, maintainer and administrator, represented by one person, i.e. the domain and system specialist who knows the software, its FRs and QRs, how it is to be used and how it had been used during its operation before.

These two persons conducted three interviews of about two hours each. The first interview specified the FRs resulting in 14 Use Cases. The second interview produced unstructured QRs, which afterwards were put together to a first Misuse Tree by the MOQARE specialist and complemented by results of the analysis of the software documentation and the software itself. Then, a third interview validated and completed the requirements in the Misuse Tree and was guided by the four steps presented in section 4. The Misuse Tree was used to structure the requirements and served as a discussion guideline for the iterated interviews. It summarized the results

of the former interviews and analyses and defined the questions for the following requirements elicitation. Altogether we gathered 50 countermeasures.

For the sake of simplicity, Figure 2 shows only one section of the whole Misuse Tree for the Uveitis Database. In the following, we describe the most important results:

1.) On the business level, we found two *business goals*, both being equally important for the Uveitis Database: “good therapy for patients at optimal cost” and “valid scientific studies”. The first of them has several factors: A good therapy is based on all the examination results, former medications and their effects. Performing all possible examinations is neither effective, nor time-efficient, nor cost-efficient. To identify the best order of examinations and medications, leading to a high quality therapy at the lowest possible cost, the Uveitis Database has to contribute to the management and the supply of correct, complete and up-to-date data of patients, their examination results, medications and diagnosis.

Both business goals - “good therapy for patients at optimal cost” and “valid scientific studies” - are threatened by the quality deficiency “wrong, missing or outdated data” as well as by the “disclosure of individual patient data”. With respect to therapy, wrong, missing and outdated data can lead to the business damage ”suboptimal therapy”, which causes avoidable cost and - in the worst case – blindness of a patient, who otherwise (i.e. with the right data being available) could have been cured or at least the illness could have been slowed down. With respect to the scientific study, wrong, missing and outdated data lead to wrong study results. Consequently, the study would not be accepted by the scientific community, and the institute would lose money, time and fame. Individual data of patients are subject to strict data protection laws in Germany. Although they must be available for the persons involved in the therapy, and can be used and published in anonymous form in scientific publications, the individual patient data must not be disclosed to unauthorized persons. Such a breach of the data protection laws would lead to litigation with all its consequences (cost, negative publicity, etc.).

The quality deficiency “wrong, missing or outdated data” leads us to the QA “integrity” and the asset “data”, i.e. the quality goal “data + integrity”. The quality deficiency “disclosure of individual patient data” leads to the quality goal “individual patient data + privacy”. On this top level of the analysis, it seems that only mere data was important in the case study, but later-on the MOQARE analysis will show, that if we want to control the privacy and integrity of data, the whole process of data input, processing and output as well as the software development process has to be controlled.

2.) For each quality goal, several threats, misusers and vulnerabilities could be derived, using the checklists as a basis for creativity. In the Misuse Tree, we group a threat with its misuser and vulnerability in a Misuse Case. To keep the size and complexity of the Misuse Tree at a minimum, we summarized several vulnerabilities, which lead to similar threats within one Misuse Case.

3.) For each Misuse Case, we derived countermeasures, also using the checklists.

4.) Re-start of the cycle: As was expected, many countermeasures were new quality goals, like the “usability of user interfaces”. Such a quality goal needs a new analysis, re-starting at step 2. Usability for instance is a countermeasure for user

errors which threaten the quality goal “data + integrity” and also “individual patient data + privacy”. In fact, pilot use had shown that user errors are a major source of invalid data. Hence, further QAs were considered which do not support the business goals directly, but indirectly.

The complete Misuse Tree produced by this case study is too big to be presented here. It contained 8 layers (i.e., two iterations). On the first level, there were two quality goals, ten threats and 35 countermeasures. Of these countermeasures, 13 were quality goals and three of these were analysed further, leading to ten more threats and 15 countermeasures. The other quality goals were not analysed, because other stakeholders are responsible for their satisfaction or they are already being sufficiently supported by the standard software in use. The 15 quality goals belonged to all six categories of ISO 9126: Availability and usability appeared twice and usability three times, each time referring to another asset. Integrity, privacy, recoverability, learnability, maintainability, portability, time-efficiency and fault tolerance appeared once. All QRs found during the case study could be elicited or justified using Misuse Cases, and we expect this to be so in general.

Figure 2 shows an extract of this Misuse Tree, focused on the quality goals concerning usability. They were derived like this: Quality goal “data + integrity” is threatened by several Misuse Cases, one of them being “unintentional data corruption by user error”. The new quality goal “usability of interfaces where users enter data” is one countermeasure out of sixteen for this threat. Other potential Misuse Cases to this quality goal were: “intentional corruption of data by intruders”, “use of data format which is inadequate for domain data”, “wrong or inaccurate calculation”, “intentional corruption of data by users”, “data corruption by system error/ break-down”.

The quality goal “individual patient data + privacy” is threatened by the Misuses “intentional disclosure by intruder”, “intentional disclosure by user”, “unintentional disclosure by user”, “unintentional disclosure by developer during software enhancement”, “developers see confidential data during software tests with productive data”. The probability of the threat “unintentional disclosure by user” is reduced by the quality goal “usability of printout functionality” and “usability/ learnability of interfaces”, but also by the definition of a security policy, the training of users concerning this policy and users signing data protection commitments.

Concerning data integrity, a lot of threats could be identified, especially as the system has already been in operative use, and former user errors could be analyzed. Most of them were of general types as they occur in each system, like typing errors (countermeasures: value lists, automated reading of data via card reader from insurance card, plausibility check, …), incomplete input (countermeasure: obligatory fields which enforce input), creation of an empty data set (countermeasures: confirmation message at data set creation, obligatory fields, data cleansing by a script searching for empty data sets), impatience of the user (countermeasure: time-efficiency of the system). But also system-specific vulnerabilities were identified, like the similarity of the search and input interfaces, or the possibility to save an empty data set.

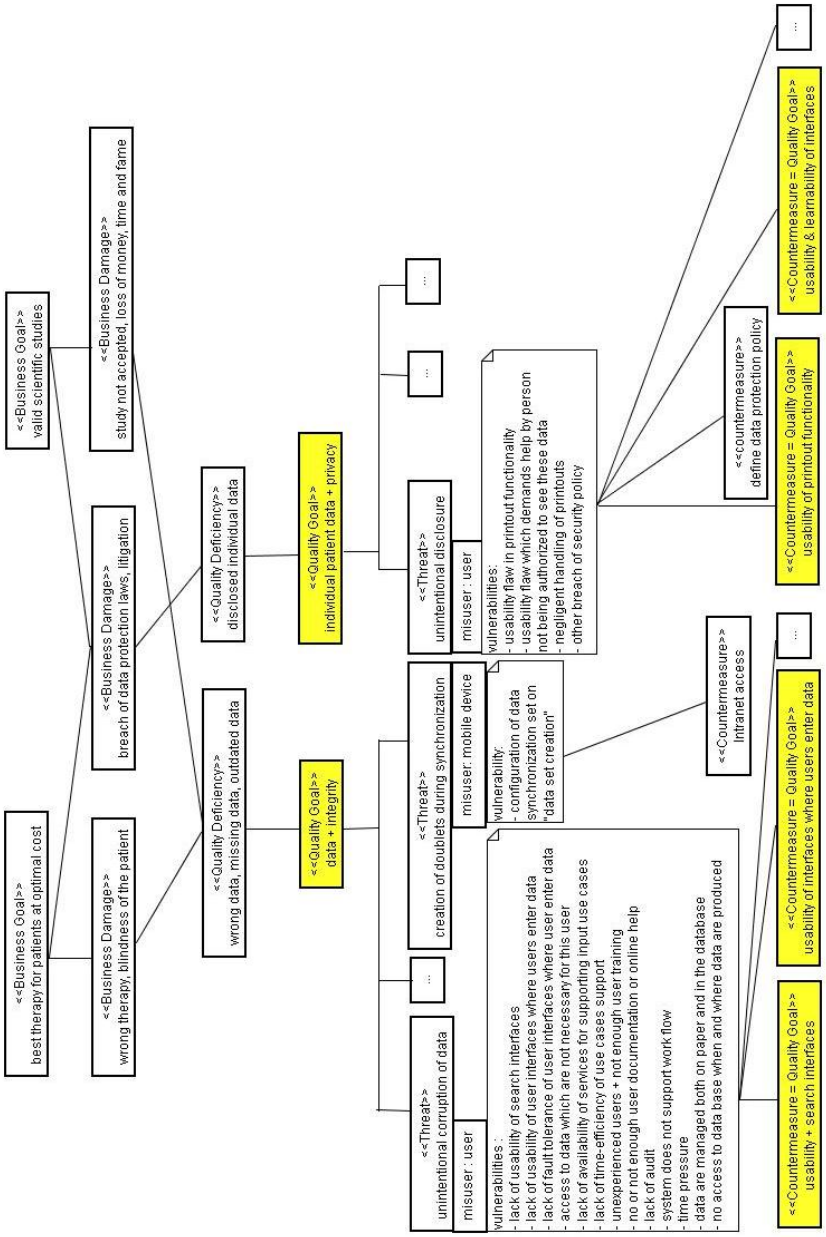

Figure 2: Part of the Misuse Tree for the Uveitis Database, focused on usability aspects. Rectangles containing three dots […] indicate where further elements have been left out.

6 Lessons Learned

What did we learn from the case study? MOQARE supports the systematic investigation of QRs. The requirements elicitation was well guided by the four steps of concept elicitation, the Misuse Tree and by the checklists which help to ask result-oriented questions.

The Misuse Cases seemed to be easy to understand for the domain experts (developer and ophthalmologist). The hierarchical graphical presentation of the Misuse Tree gives an overview of the requirements (which can be documented in more detail in another document, if required) and visualizes relationships among goals, requirements and Misuse Cases. Therefore, it helped to structure the interviews. A simple graphic UML tool is sufficient to navigate within the tree, to update its content and to print the tree or sections of it.

Our iterative requirements elicitation process was supported by the Misuse Tree: At each iteration, first the Misuse Tree resulting from the former iteration was reviewed and then used as an interview guide for further requirements elicitation. Then a branch was chosen where the interview was to continue to bring forth new results. In this way, the tree structure helped to structure the elicitation process, but also allowed us to add spontaneous ideas at the right position.

The checklists were helpful in avoiding the concentration on only a few QAs, types of threats or misusers, as happens easily in unstructured discussions. In the case study, QAs that emerged during the discussion thanks to the checklist analysis were recoverability, maintainability and portability, i.e. those QAs which refer to growth scenarios.

The iterative assessment of requirements, proceeding from a general level down to more and more details, combined with document analysis and analysis of the software, was an efficient way of gathering the needed information with a minimum effort for the domain expert and a good way of avoiding misunderstandings.

Not only did we gather 50 countermeasures, but also analysed their motivations, starting from business goals. This selection of the most important countermeasures also signified to decide that the other possible countermeasures on our checklists are less relevant.

The Misuse Tree represents several different QAs and the relationships among them. For example, data integrity depends both on security and on usability, and usability also depends on time-efficiency. Therefore, it is reasonable to consider all QAs in an integrated approach, not ignoring but integrating the expertise of the HCI (Human Computer Interaction) community concerning usability or of the security and reliability community, etc…

As the analysis aimed to find system specific requirements, the wording of the items in the checklists was too general. A domain-specific wording should always be preferred. For instance, “user” should be replaced by a specific role like “nurse”, and the quality goal “usability of interfaces where users enter data” must be concretized by naming these interfaces e.g. “usability of user interface ‘create and edit patient data’”.

Several quality goals which emerged during the case study could be satisfied by generally known solutions, not specific to the Uveitis Database. An example of this is the intrusion of a hacker. As countermeasures, intrusion detection and all measures

which prevent intrusion or at least make it more difficult, can be proposed. Such solutions are known and standard products are on the market, like intrusion detection software. The corresponding countermeasures can be found in our general checklists and the specialized literature. The hospital´s specialists already apply a whole bundle of countermeasures to prevent intrusions. Therefore, we stopped our analysis here. Therefore, we focused on the analysis of 5 quality goals out of the 15. Otherwise, more layers would have been included in the Misuse Tree.

Some of the countermeasures might look trivial and are common good practice in software engineering like “compliance to known usability rules” or “good testing”. But if our aim is a comprehensive description of all important requirements, this is a good result and shows that the checklists also help to explicitly name the “tacit assumptions” so much searched for by requirements engineers. We think that the logic behind this observation is as follows: These trivial requirements are considered to be trivial, because they prevent misuses that are relevant to practically all software systems. Nevertheless, they are important for protecting a business goal.

Not only requirements referring to the software were discovered, but also requirements and constraints on the software development process or the project. This not only happened because we explicitly included them from the beginning (e.g. in the checklists, as the security literature does), but also because they strongly affect quality. Software quality is the result of good software engineering, and therefore the analysis would not be complete without such requirements.

Some Use Cases which represent countermeasures refer to tasks like data cleansing or maintenance, i.e. to tasks which might easily be forgotten by the stakeholders during the requirements analysis. In MOQARE, you are reminded to include them in the description of requirements, as data cleansing and maintenance in fact are tasks one needs to perform in the system. These tasks not only improve the system quality, but are necessary to sustain the quality level of the new system. In a dynamic environment, the quality of a system can be expected to decrease, if it is not maintained.

Adhering to the definitions of the concepts is important for the completeness of the results. However, this is difficult and in case studies this goal cannot be achieved neatly by untutored stakeholders. Vulnerabilities and threats, causes and consequences, are easily being mixed up. Therefore, MOQARE requires a method specialist to translate the stakeholders´ unstructured requirements into a Misuse Tree.

We started with 14 Use Cases, but the number of all Misuse Cases amounts to several dozens. We cannot give the exact number, as we stopped our analysis where the discussion started to become too general and where standard solutions are known (see above). Only think of the large number of possible Misuse Cases for intrusion into the network.

Several more case studies were performed ([21]-[24] and unpublished) and they support these lessons learned.

7 Related Work

When developing MOQARE, we strongly relied on other publications which describe methods for QRs elicitation. In this section, we state our sources and other related literature. Researching the relevant literature, we identified the concepts commonly used and based on these concepts, we developed the comprehensive and clearly defined conceptual framework of MOQARE. Tables 2 to 4 show that our sources share similar concepts, but MOQARE is the first one to cover all of them, providing a way of modelling QRs in more detail and more systematically than other methods.

Misuse Cases have been successfully used to elicit and operationalize the QR "security". We follow a suggestion of Alexander [25]: "There is scope for further work applying Misuse Cases to elicit Usability requirements." In [26], Alexander applied the Misuse Cases to reliability, maintainability and portability. Firesmith [27] highlights the similarities of safety, security and survivability. We went one step further and developed a systematic method to apply Misuse Cases to all other QAs. Misuse Cases have not been applied to all QA before in an equal way. And when Misuse Cases were applied to particular other QA like reliability, necessary generalizations of the concept definitions were not discussed.

The concept of Misuse Cases has a short history. 1999, McDermott and Fox [28] introduced the term 'Abuse Case' for eliciting security requirements. Sindre and Opdahl [2],[11] explicitly call them Misuse Cases. Allenby and Kelly [29] describe a similar method of eliciting and analyzing safety requirements for aero-engines using what they call 'Use Cases'. The concept of Misuse Cases has been used successfully since, and several field reports are available [7][25][30][31]. However, there are few systematic methods of deriving them other than enhanced UML Use Case diagrams and templates. Usually they are used intuitively.

We soon found out that generalizations of the definitions of the Misuse Case concepts were necessary to apply Misuse Cases to all QAs defined by ISO 9126. Therefore, we searched in Misuse Cases, risk analysis, security, reliability, QR and architecture analysis contexts for further concepts and ways of specifying QRs. We were especially interested in approaches describing and analyzing QRs by identifying threats, Misuse Cases, and everything which is NOT to happen, as we expected that this approach helps to investigate QRs and to complement the requirements. The concepts of assets, vulnerability and threats are implicitly used everywhere in the area of security assessment, see for example [31] and [32], but these concepts are often not clearly defined and even mixed, because of a lack of differentiation. These sources all have in common that they search for logical and causal relationships of QRs with other system concepts. In the area of security related requirements, we looked at the following work:

The abuse case model of McDermott and Fox [28]

Misuse Case templates of Sindre and Opdahl [2], [11]

The attack patterns of Moore, Ellison and Linger [33]

Lin et al. [34] apply problem frames to identify security requirements

Van Lamsweerde at al. [19] develop an extension of the KAOS framework which considers intruder anti-goals against system goals, additionally to the goal-anchored trees modeling FR

Liu, Yu and Mylopoulos [35] develop a framework based on i*

Firesmith´s templates for reusable security requirements [7],[13],[14]

Object Management Group [36] integrate the risk assessment concepts into the UML standard and enhance the standard by using five sub-models

Blakley, Heath, and other members of The Open Group Security Forum [37] published a catalogue of security patterns and a generic method using them to design a system architecture

As can be seen in Tables 2 and 3, they all use different terminology and do not cover all MOQARE concepts in full generality. Not only are various terms used for the same concept, but the same word has different meanings. For instance, the term 'threat' is used by other authors as well, but its definition is not clear. Firesmith [7] uses this term to describe the anti-goal of a misuser and lists the "security threats" "theft, vandalism, fraud, unauthorized disclosure, destruction, extortion, espionage, trespass" as categories, which are rather consequences of security problems than their causes. Others [32] mix misusers, forces of nature, motivation of the misuse, vulnerabilities or the consequences of Misuse Cases in the same list. The same holds true for the following general QR methods listed in Table 4:

ATAM (Architecture Tradeoff Analysis Method) [5][38], evaluates several architectural styles or solutions with each other, using QRs as evaluation criteria

The EMPRESS Quality Models [10][20] link QAs to means for satisfying them and to metrics for measuring quality, in a tree structure

Sutcliffe and Minocha´s scenario templates for process guidance in early exploration and validation of QRs [12]

The NFR Framework also derives requirements from goals: Chung et al. [9] decompose QAs (called "softgoals") in their sub-goals and derive 'operationalizations'. They also document contributions of means to the softgoal satisficing, priorities, decisions and their rationale, within the softgoal graph. The softgoal networks are also used as a means of cataloguing QRs for reuse [39]. In some works, the researchers using softgoal graphs also distinguish between technical objectives and business objectives [40], [41]. Countermeasures sometimes also are expressed as scenarios [42]. But the NFR Framework does not use any negative elements (threats, vulnerabilities and misuse cases), i.e. such concepts which describe what is not wanted (see Table 4). We think that these negative elements make MOQARE more intuitive and they better document the motivations of requirements.

Not presented in the tables is FMEA (Failure Mode and Effect Analysis) [43]. It is a well-established method for deriving actions (i.e. requirements respectively countermeasures) via failures (threats). These failures have causes (vulnerabilities) and effects (quality deficiencies), and these are all gathered in templates. This method analyses both systems and processes. It is implicitly focused on functionality and reliability requirements. We think that mainly the graphical presentation of requirements in the Misuse Tree is an advantage over FMEA.

We analysed more references without discovering further concepts. Therefore, we merely used their extensive lists of threats, vulnerabilities and countermeasures to compile our checklists [31], [32], [44-50].

Table 2: Terminologies used in five references on security requirements

MOQARE	Abuse Case model of McDermott and Fox [28]	Misuse Case templates of Sindre and Opdahl [2]	Moore, Ellison and Linger [33]	Van Lamsweerde et al. [19]	Liu, Yu and Mylopoulos [35]
Asset	Use Case	---	---	---	---
QA	(Security)	(Security)	---	Goal	Security, privacy
Vulnerability	---	(assumptions, preconditions)	Precondition	Vulnerability	Dependencies lead to vulnera- bilities
Threat	Exploit	Threat	Attack	(intentional and non-intentional) Obstacle	Threat
Consequence (=quality deficiency or business damage)	Harm	Post-condition	Post-condition	---	---
Misuse Case	Abuse Case	Misuse Case, course of events	Attack pattern steps	Attack (intentional & unintentional)	---
Misuser	Actor	Misactor, stakeholder	Attacker	Attacker	Attacker (as subset of Actors)
Misuser Attribute	Resources, skills, objectives	Misuser profile	Attack goal, precondition	Anti-goal	Malicious intent
Counter-measure	Use Case	---		Security Requirements	Counter-measure

Table 3: Terminologies used in five references on security requirements

MOQARE	Misuse Cases [13]	Template of Firesmith [7]	Template of Firesmith [14]	UML enhancement [36]	Blakley, Heath, et al. [37]
Asset	Assets and services	(Vulnerable) Asset	Asset	Asset = target of evaluation	Resources, critical

					components, protected system instance
QA	---	(Security)	Security goal, security quality factor	QoS Category, asset value	Availability + security
Vulnerability	---	Vulnerability	Vulnerability	Vulnerability, weakness	(Applicability)
Threat	Threat	Attack	Threat, attack	Unwanted incident	---
Consequence	---	Negative outcomes	negative impacts, harm	Consequence	Consequences
Misuse Case	Misuse Case	Situation	---	Threat	---
Misuser	Misuser	Attacker	Attacker	Threat Agent	Actor, attacker
Misuser Attribute	---	(Anti-)Goal	---	---	---
Counter-measure	Security requirements, security mechanisms	Requirement, quality sub-factors	Requirement, security sub-factor, security criterion	Treatment	Protected System pattern, policy

Table 4: Terminologies used in five references on QRs

MOQARE	ATAM [5]	EMPRESS Quality Model [10], [20]	Template of Sutcliffe & Minocha [12]	NFR Framework: Chung, Cysneiros, Leite, Mylopoulos, Nixon, Yu [9], [39], [42]	Lin et al. [34]
Asset	(Trade-off point)	---	---	Softgoal topic	Asset
QA	Quality Attribute (Response)	Quality Attribute	NFR (non-functional requirement)	Softgoal type	Security (confidentiality, integrity, availability)
Vulnerability	Sensitivity Point, (Architec-tural) Risk	---	---	---	Vulnerability
Threat	---	---	Expected failure	---	Threat
Consequence	---	Damage	Damage	Contribution	Anti-

					requirements
Misuse Case	Scenario	---	Scenario	---	
Misuser	Stakeholder (customer, maintainer and developer)	Stakeholder (customer and developer)	Agent	---	Attacker
Misuser Attribute	Stimulus	---	Motivation	---	---
Counter-measure	Response	Means	Counter-measure	Sub-goal, operationa-lization	Security requirements

Most of these sources do not offer a systematic process for deriving requirements. Concepts are not clearly defined or not general enough to apply them to all types of QAs. Our conceptual framework includes the total of all concepts typically used in literature. The MOQARE process which is described in section 4, builds on the primitive process commonly applied by all authors using threats or Misuse Cases: Starting from vague quality goals or QAs, Misuse Cases are developed and then countermeasures are derived (steps 2 and 3). What is new is the derivation of the quality goals from business goals (step 1) or from other quality goals, which leads to recursion (step 4). We think it is important to differentiate between quality goal (asset + QA) and countermeasure to judge whether the objective of deriving realizable requirements is reached. We are the first to do so.

In this section, we exclusively relied to references which describe methods for QR elicitation and specification, because this is the focus of our present work. Of course, there are many other aspects of quality which we did not treat here, like the trade-off among contradicting quality requirements, processes of quality management in general (e.g. Total Quality Management [51]), organizational measures or programming paradigms for improving quality, or metrics for measuring quality.

8 Conclusions and Future Work

This paper presents a conceptual framework and the method MOQARE for a systematic elicitation and graphical documentation of QRs. MOQARE is based on the concept of Misuse Cases and on reusable lists of QAs, threats, vulnerabilities and countermeasures. Its result is a so-called Misuse Tree.

MOQARE is demonstrated by applying it to a clinical database in a case study. We think that an analysis of QRs in terms of quality goals, Misuse Cases and countermeasures helps to complement software and project requirements. To support a complete view on system quality, we consider not only end-users, but also system administrators, maintainers and intruders of the system. We take account of normal use, growth and exploratory scenarios. Not only requirements on software and hardware are derived, but also on their development, use and maintenance. An important merit of our approach is its general applicability to all QAs.

MOQARE has been used successfully in a case study for software that already exists and which is in operative use. We also applied MOQARE to systems before their implementation. In these cases, the countermeasures found were more general and could be concretized further during the system design. MOQARE was able to derive all types of requirements via Misuse Cases, and we expect that this is generally the case. The Misuse Case scenarios are intuitive and allow to predict – more specifically than other methods - which of the potential misuses are the most relevant ones for a system. In our case study, this prediction was facilitated by former operation experience. When analysing a system before its implementation, it is useful to adapt experience from similar systems.

Some of the requirements found by MOQARE in the form of countermeasures are further FRs as well as QRs on FRs. Therefore, it makes sense to integrate the results of MOQARE with FRs. Such integration will be a good basis for the trade-off between conflicting requirements, for the design, implementation and test of the system. MOQARE has been successfully integrated with a method for FRs elicitation and documentation.

We did not yet consider here the threats and side-effects caused by the implementation of countermeasures. During the case study, one such example was discussed. As data can be input on a mobile device, from time to time, data on the mobile device and on the Uveitis Database must be synchronized. Usually, time stamps help the system to identify those data which are more recent. But when data concerning the same patient have been changed on both systems, conflict happens. The configuration of the synchronization can be on “overwrite”. Then, the data in the Uveitis Database are assumed to have higher priority and data on the mobile device are overwritten. But data entered on the mobile device can then be lost. The countermeasure against this threat is the configuration “data set creation”, which creates a new data set on the Uveitis Database (see Figure 2). But this can lead to doublets. So, both configurations threaten the quality goal “data integrity”. (As is shown in Figure 2, the third and best solution has been chosen as countermeasure: to provide Intranet access to the Uveitis Database from all work places and thus to avoid data conflict, as all users enter new data on the database directly.) Such threats provoked by a countermeasure are important during the trade-off analysis of requirements, but not during elicitation. This also applies to other relationships between countermeasures. For example, “user training on security policy” and “define security policy” depend on each other in several ways: Training on the policy can only be done after such a policy has been defined. Training was judged to be a more effective countermeasure, and the mere definition of a policy does not improve much by itself. Such dependencies are treated within the method ICRAD [52], which integrates MOQARE with requirements conflict trade-off and architectural design. These activities are strongly interrelated, as was motivated by Paech et al. [53].

MOQARE as well as its integration with FRs and architectural design needs tool support. A simple UML tool allows us to edit a Misuse Tree as a class diagram or component diagram. We are currently developing a better, web-based tool support which also allows the integration of the QR analysis into the FR description. This tool is based on Sysiphus [54].

Acknowledgements

MOQARE is the result of the research project SIKOSA, which was funded by the Ministry of Science, Research and Art of Baden-Württemberg, Germany (Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg).

We would like to thank Professor Matthias Becker and Damian Plaza of the Interdisciplinary Uveitis Center Heidelberg for their friendly and dedicated cooperation on the case study. We also thank Maike Gilliot and Lutz Lowis from the University of Freiburg and our colleagues from Heidelberg for many constructive discussions.

References

1. Paech B, Kerkow D. (2004) Non-functional Requirements Engineering: Quality is Essential. In Regnell B, Kamsties E, Gervasi V (eds). Proceedings of the 10th Intl. Workshop on Requirements Engineering: Foundation of Software Quality - REFSQ 04, Essener Informatik Beiträge Bd 9, Essen/ Germany, pp 237-250.
2. Sindre G, Opdahl AL (2001) Templates for Misuse Case Description. In Proceedings of the 7th Intl. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 01, Essener Informatik Beiträge Bd.6, Essen/ Germany, pp. 125-136.
3. Sindre G, Firesmith DG, Opdahl AL (2003) A Reuse Based Approach to Determining Security Requirements. In Proceedings of the 9th Intl. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 03,,Essener Informatik Beiträge Bd.8, Essen/ Germany, 37-46.
4. Herrmann A, Paech B (2005) Quality Misuse. In Kamsties E, Gervasi V, Sawyer P (eds). Proceedings of the 11th Intl. Workshop on Requirements Engineering: Foundation of Software Quality - REFSQ 05, Essener Informatik Beiträge Bd. 10, Essen/ Germany, pp 193-199.
5. Kazman R, Klein M, Clements P (2000) ATAM: Method for Architecture Evaluation. CMU/SEI-2000-TR-004, Software Eng. Inst., Carnegie Mellon University.
6. Wiegers KE (2002) Success Criteria Breed Success, The Rational Edge, 2(2)
7. Firesmith DG (2003) Analyzing and Specifying Reusable Security Requirements. Requirements for High Assurance Systems (RHAS) Workshop.
8. International Standard ISO/IEC 9126. Information technology - Software product evaluation - Quality characteristics and guidelines for their use.
9. Chung L, Nixon BA, Yu E, Mylopoulos J (2000) Non-Functional Requirements in Software Engineering. Kluwer Acadamic Publishers.
10. Dörr J, Punter T, Bayer J, Kerkow D, Kolb R, Koenig T, Olsson T, Trendowicz A (2004) Quality Models for Non-functional Requirements. IESE-Report Nr. 010.04/E.
11. Sindre G, Opdahl AL (2000) Eliciting Security Requirements by Misuse Cases. TOOLS Pacific 2000: 120-131.
12. Sutcliffe A, Minocha S (1998) Scenario-based Analysis of Non-Functional Requirements. In Dubois E, Opdahl AL, Pohl K (eds.). Proceedings of the Fourth Intl. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 98, Presses universitaires de Namur, Namur/Belgium, 219-234.
13. Firesmith DG (2003) Security Use Cases. Journal of Object Technology 2(3): 53-64.

14. Firesmith DG (2004) Specifying Reusable Security Requirements. Journal of Object Technology 3(1): 61-75.
15. Herrmann A, Paech B (2005) Software Quality by Misuse Analysis. Technical Report SWEHD-TR-2005-01 (University of Heidelberg), http://www-swe.informatik.uni-heidelberg.de/ research/publications/reports.htm.
16. BSI (Bundesamt für Sicherheit in der Informationstechnik = German Ministery for Security in Information Technology) (2004) IT-Grundschutzhandbuch. http://www.bsi.bund.de/gshb/deutsch/g/g01.html.
17. Nakajo T, Kume H (1991) A Case History Analysis of Software Error Cause-Effect Relationships. IEEE Transactions on Software Engineering 17 (8): 830-838.
18. Lutz RR (1993) Analysing software requirements errors in safety-critical embedded systems. Requirements Engineering Conference, IEEE Computer Society Press, Silver Spring: 126-133.
19. van Lamsweerde A, Brohez S, De Landtsheer R, Janssens D (2003) From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering. RHAS'03 Workshop: 49-56.
20. Dörr J, Kerkow D, von Knethen A, Paech B (2003) Eliciting Efficiency Requirements with Use Cases. In Proceedings of the 9th Intl. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 03, Essener Informatik Beiträge Bd.8, Essen/ Germany, 37-46.
21. Herrmann A, Rückert J, Paech B (2006) Exploring the Interoperability of Web Servic**es** using MOQARE. IS-TSPQ Workshop "First International Workshop on Interoperability Solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems", 21th March 2006 in Bordeaux
22. Herrmann A (2007) Priorisierung von Qualitätsanforderungen auf der Basis von Risikoabschätzungen. Software & Systems Quality Conferences International 2007, 27th April 2007, Düsseldorf, Germany.
23. Herrmann A, Kerkow D, Doerr J (2007) Exploring the Characteristics of NFR Methods – a Dialogue about two Approaches. In Proceedings of the 13th Intl. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 07, to be published
24. Weiß D, Kaack J, Kirn S, Gilliot M, Lowis L, Müller G, Herrmann A, Binnig C, Illes T, Paech B, Kossmann D (2007) Die SIKOSA-Methodik – Unterstützung der industriellen Softwareproduktion durch methodisch integrierte Softwareentwicklungsprozesse. Wirtschaftsinformatik 49(3): 188-198.
25. Alexander I (2002) Initial Industrial Experience of Misuse Cases. Requirements Engineering Conference: 61-68.
26. Alexander I (2003) Misuse Cases: Use Cases with hostile intent. IEEE Software 20 (1): 58-66.
27. Firesmith DG (2003) Common Concepts Underlying Safety, Security, and Survivability Engineering. Technical Note CMU/SEI-2003-TN-033.
28. McDermott J, Fox C (1999) Using Abuse Case Models for Security Requirements Analysis. 15th Annual Computer Security Applications Conference ACSAC: 55-65.
29. Allenby K, Kelly T (2001) Deriving Safety Requirements Using Scenarios. Requirements Engineering Conference: 228-235.
30. Aagedal JÖ, den Braber F, Dimitrakos T, Gran BA, Raptis D, Stölen K (2002) Model-based Risk Assessment to Improve Enterprise Security. 5th International EDOC Conference: 51-62.
31. Moffett JD, Haley CB, Nuseibeh B (2004) Core Security Requirements Artefacts. Technical Report No 2004/23, Department of Computing, The Open University, UK.

32. Computer Security Resource Center (CSRC). Common Criteria, Version 2.1. http://csrc.nist.gov/cc/.
33. Moore AP, Ellison RJ, Linger RC (2001) Attack Modeling for Information Security and Survivability. Technical Note CMU/SEI-2001-TN-001.
34. Lin L, Nuseibeh BA, Ince DC, Jackson M, Moffett JD (2003) Analysing Security Threats and Vulnerabilities Using Abuse Frames. Technical Report No: 2003/ 10, Department of Computing, The Open University, United Kingdom.
35. Liu L, Yu E, Mylopoulos J (2003) Security and Privacy Requirements Analysis with a Social Setting. Requirements Engineering Conference: 151-161.
36. Object Management Group (2004) UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. http://www.omg.org/docs/ptc/04-09-01.pdf.
37. Blakley B, Heath C, and members of The Open Group Security Forum. Security Design Patterns. http://www.opengroup.org/onlinepubs/9299969899/toc.pdf.
38. Kazman R, Klein M, Barbacci M, Longstaff T, Lipson H, Carriere SJ (1998) The Architecture Tradeoff Analysis Method. Software Engineering Institute, Technical Report CMU/SEI-98-TR-008.
39. Cysneiros LM, Yu E, Leite JCSP (2003) Cataloguing Non-Functional Requirements as Softgoal Networks. Proc. of Requirements Engineering for Adaptable Architectures, at RE´03:13-20.
40. Mylopoulos J, Chung L, Liao SSY, Wang H, Yu E (2001) Exploring Alternatives During Requirements Analysis. IEEE Software 18(1): 92-96.
41. Liu L, Yu E (2001) From requirements to architectural design - Using goals and scenarios. Workshop STRAW, ICSE 2001, http://www.cin.ufpe.br/~straw01/.
42. Cysneiros LM, Leite JCSP (2001) Driving Non-Functional Requirements to Use cases and Scenarios. Proc. of XV Brazilian Symposium on Software Engineering: 7-20.
43. Stamatis DH (2003) Failure Mode and Effect Analysis - FMEA from Theory to Execution. American Society for Quality Press, Milwauki, USA
44. BSI (Bundesamt für Sicherheit in der Informationstechnik = German Ministery for Security in Information Technology) (1989) IT security criteria. http://www.bsi.bund.de/zertifiz/itkrit/itgruene.pdf.
45. BSI (Bundesamt für Sicherheit in der Informationstechnik = German Ministery for Security in Information Technology) (1991) Information Technology Security Evaluation Criteria. http://www.bsi.bund.de/zertifiz/itkrit/itsec-en.pdf.
46. Landwehr CE, Bull AR, McDermott JP, Choi WS (1994) A taxonomy of computer program security flaws, with examples. ACM Computing Surveys 26(3): 211-254.
47. Aslam T (1995) A taxonomy of security faults in the Unix operating system. Master´s thesis, Purdue University.
48. Anton AI, Earp JB, Reese A (2002) Analyzing Website Privacy Requirements Using a Privacy Goal Taxonomy. Requirements Engineering Conference: 23-31.
49. Richardson R (2003) 2003 CSI/FBI Computer Crime and Security Survey. Computer Security Institute, http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2003.pdf.
50. Killourhy KS, Maxion RA, Tan KMC (2004) A Defense-Centric Taxonomy Based on Attack Manifestations. International Conference on Dependable Systems & Networks: 102-111.
51. Feigenbaum AV (1961) Total quality control - engineering and management. McGraw-Hill, New York
52. Herrmann A, Paech B, Plaza D (2006) ICRAD: An Integrated Process for Requirements Conflict Solution and Architectural Design. IJSEKE 16(6): 1-34.

Formatiert: Deutsch (Deutschland)

53. Paech B, Dutoit A, Kerkow D, von Knethen A (2002) Functional requirements, non-functional requirements and architecture specification cannot be separated - A position paper. In Proceedings of the 8th Intl. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 02, Essener Informatik Beiträge Bd.8, Essen/ Germany, pp. 102-107.
54. Sysiphus http://sysiphus.in.tum.de/, 2007