Case study in section 6 was removed because it is confidential

Software Quality by Misuse Analysis

Working Paper 2005-AH-1

Authors:
Andrea Herrmann
Barbara Paech

Version 1.5
9th Jan 2007

Case study in section 6 was removed because it is confidential

Case study in section 6 was removed because it is confidential

The „Software Systems Engineering“ Group is part of the Institute of Mathematics and Computer Sciences of the Ruprecht-Karls-Universität Heidelberg. It is managed by

Prof. Dr. Barbara Paech
Institut für Informatik
Neuenheimer Feld 348
69120 Heidelberg
Germany

paech@informatik.uni-heidelberg.de
http://www-swe.informatik.uni-heidelberg.de/

Case study in section 6 was removed because it is confidential

Case study in section 6 was removed because it is confidential

Abstract:

The research project SIKOSA is funded by the Ministery for Science, Research and Art of Baden-Württemberg, Germany (Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg). We also want to thank Damian Plaza who spent many hours on the case study.

Within the research project SIKOSA and its work package „Requirements“ a method for requirements elicitation and documentation has been developed which derives all types of non-functional requirements by using misuse cases, not only security requirements. This “Misuse-oriented Quality Requirements Engineering” method (short: MOQARE) has been integrated with the elicitation and documentation of functional requirements.

There are several methods for the derivation and analysis of detailed non-functional requirements. These methods often are designed for a restricted field of application, e.g. misuse cases for top-down derivation of requirements detailing the quality attribute “security”, or ATAM for evaluating given architectural alternatives. In this work, we apply misuse cases to any other quality attribute (e.g. usability, maintainability) to develop a method for deriving detailed non-functional and functional requirements from any quality attribute. Doing so, we find that generalizations have to be made to the definitions of the misuse case concepts, and new concepts must be included. We applied our method successfully in a case study. It was a good tool for systematic and concrete requirements elicitation easily understood by the stakeholders and leading to realizable requirements.

By the “Misuse-oriented Quality Requirements Engineering” method (short: MOQARE), NFR were operationalized as to be realizable, and at the end, further FRs for the system were found and constraints to FRs. Therefore, it makes sense to present the results of the NFR method with the FRs in an integrated presentation. Such a presentation will be a good basis for the design, implementation and test of the system, much better than a separate presentation of FR and NFR.

This working paper is the first out of a series which will be produced in the SIKOSA research project. While here the integration of NFR and FR during the requirements elicitation is treated, later papers will investigate the interface between requirements and architectural design, requirements and test cases, and requirements and project management.

The case study which was performed with MOQARE is confidential.

1 Introduction

Elicitation of quality requirements for software systems often starts with quality goals vaguely expressed and applying to the whole system (e.g.: “The system shall be secure/ easily usable/ fast.”). However, such a requirement is neither detailed enough for being implemented by a developer nor for being tested by a software tester, not to mention its usefulness for cost estimation.

There are several methods for exploring non-functional requirements (NFR). These methods often are designed for a restricted field of application, like Misuse Cases [SO00][SO01] for detailing the QA “security”, or ATAM (=Architecture Tradeoff Analysis Method) [KKC00] for evaluating given architectural alternatives. In this work, we apply misuse cases to other quality attributes (QAs). The misuse cases in the realm of security have given the benefit of completing the view on a system considering exceptions and threats, not only the intended and successful use. To define and detail the successful use of a system and its quality, it is important to think about threats to both. Furthermore, misuses help to relate NFR to functional requirements (FR), which in our opinion must be considered “in a tightly integrated approach” [PDKV02]. Sutcliffe and Minocha [SM98] as well as Hochmüller [Hoc97] suppose that NFR express constraints on FR and design.

Our aim was to develop a misuse-based method, the so-called “Misuse-oriented Quality Requirements Engineering” method (short: MOQARE), for deriving detailed non-functional and functional requirements. This method was meant to be systematic and thorough, understandable for the stakeholders and leading to realizable, detailed requirements. Developing and testing this method, we found that generalizations to the definitions of the misuse case concepts are necessary, and new concepts must be included. We adopted elements from ATAM, the scenario templates of Sutcliffe and Minocha, and some more, and content from the softgoal graphs of Chung et al. [CNYM00], and the Quality Models of Dörr et al. [DKVP03][DPB+04].

Finally, MOQARE is integrated into the requirements engineering method TRAIN already used at our institute. We applied as well the misuse case analysis as the integrated method to perform an extensive case study, which helped to test and to improve the method.

This working paper is structured as follows: Section 2 discusses related work, section 3 presents our generalized terminology. In section 4, we present a misuse case based method for deriving requirements. Section 5 describes the integration of MOQARE into TRAIN, with a special focus on explaining how functional and non-functional requirements are integrated. In section 6, we apply the method to a case study. In section 7 we describe perspectives for further research, while section 8 summarizes the paper. Annex A contains the checklists used for MOQARE and is followed by the references.

2 Related Work

In this paper, we do not invent anything which would be totally new, but we integrate the concepts used in literature from the misuse cases, risk analysis, security, reliability, NFR or

architecture evaluation context. What *is* new is the application of misuse cases to all quality attributes other than security, and the systematic approach we developed for doing so. In this section, we refer to our sources and related literature and describe them shortly, and name the elements we adopted from them.

We not only refer to the misuse case literature itself, but also to the vast risk analysis literature. Some requirements engineering researchers also treat attacks and other concepts for operationalizing security and other non-functional requirements, and even for evaluating architectural alternatives risks have been used.

Our work is based on the Misuse Cases´ principle: Misuse Cases take the view of a misuser whose goal is to misuse the system. Misuse cases foresee his/ her behaviour and define what the system must not do or not allow. From these misuse cases new system requirements can be derived. So misuse cases help to complement the system specification.

The concept of Misuse Cases has a short history. John McDermott [MDF99] and Chris Fox introduced the term 'Abuse Case' for eliciting security requirements. Sindre and Opdahl [SO00], [SO01] explicitly call them Misuse Cases. Karen Allenby and Tim Kelly [AK01] describe a similar method for 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 experience reports are available ([ABD02], [Ale02], [Fir03b], [MHN04]).

We chose the misuse case approach as a basis for examining quality attributes because in the realm of security it has given the benefit of completing the view on a system by what must not happen, and it derives functional from non-functional requirements, which in our opinion must not be separated. We do not want to separate the specification of functional requirements (FR) from the non-functional requirements (NFR) or the architecture. They must be considered “in a tightly integrated approach” as is motivated by Paech et al. [PDKV02]: FR and NFR constrain each other, and both must be realized by the architecture (see also [SM98],[Hoc97]).

Sutcliffe and Minocha [SM98] presuppose that “NFR have a close tie with functional specification”, and that they express constraints on FR and design solutions. We adopt this view. Sutcliffe and Minocha developed scenario templates similar to misuse cases for process guidance in early exploration and validation of NFR. These templates contain, among others, the parameters “expected failure” and “damage”, “scenario description”, “agent” and “motivation”, “countermeasure”. They emphasise that as well NFR as scenarios must be tested against an existing system or some vision of a system (architecture, design or prototype). For assessing how well an NFR is satisfied in a system they decompose NFR into quality criteria and deduce metrics. The scenarios are also used as test scripts. From their terminology, we pick the term “countermeasure” as the most general expression for anything one does against a threat.

Although the misuse cases have been invented for describing security requirements, there has been the idea that they can do more. We follow a suggestion of I. Alexander [Ale02]: “There is scope for further work applying Misuse Cases to elicit Usability requirements.” In [Ale03] he himself applied the misuse cases to reliability, maintainability and portability. Firesmith [Fir03a] highlights the similarities of safety, security and survivability. We go a step further and apply misuse cases also to all the other quality attributes and, in the following two sections, develop a systematic method for doing so.

We refer to the categories of quality attributes as defined by the standard ISO 9126 [ISO91].

For treating security requirements, several requirements elicitation and documentation methods have been adapted by considering misuses using various different names and concepts.

A. van Lamsweerde and L. Willemet [LW98] in their paper intertwining goal-based and scenario-based RE regard "positive" and "negative" scenarios, i.e. desired and undesirable behaviour. They define obstacles to be working against goals.
Lamsweerde et al. in an extension of the KAOS framework consider intruder anti-goals against system goals [LBDJ03]. For identifying security requirements they also part from attacks. Goals are operationalized into specifications of operations to achieve them. These goal-anchored trees are used to describe unintentional attacks, while intentional ones are modelled by anti-goal trees. Root anti-goals are obtained by the negation of confidentiality, privacy, integrity and availability, safety, security, fault-tolerance and survivability. The anti-goals are also called "obstacles to requirements achievement". Attackers set up obstacles intentionally to break security goals. "Attack trees are derived systematically through anti-goal refinement until leaf nodes are reached that are software vulnerabilities observable by the attacker or anti-requirements implementable by this attacker." They distinguish between functional goals, non-functional goals and domain properties. In our work, we will derive a similar tree.
Liu, Yu and Mylopoulos in one of their papers [LYM03] analyse security and privacy requirements within a methological framework based on i*. They start with an actor analysis and from the actors derive attackers. Actors also lead to goals/ tasks and dependencies between actors, while attackers are attributed a malicious intent. They postulate that dependencies lead to vulnerabilities/ threats. From these they derive attacking measures and countermeasures. In a second step they rank design alternatives by a goal-based evaluation, according to the contribution of design alternatives to the softgoals of the system. We do not start with the actors, but with the assets to be protected.
The Object Management Group did integrate the risk assessment concepts into the UML Standard and enhanced this standard accordingly [OMG04]. The most important concepts are, presented in five submodels:
1.) SWOT analysis: strengths, weaknesses, opportunities, and threats
2.) context: stakeholders, assets (also called target of evaluation), asset value, policy
3.) unwanted incidents: agent, scenario, threats (threat = threat agent + threat scenario) + vulnerabilities, reduction of asset value, link to other unwanted incidents
4.) risks: risk = unwanted incident + consequence + frequency, risk value = loss of asset value, link to other risks
5.) treatments: ways of treating the system (for reducing risk), treatment effect, treatment evaluation, risk reduction, reduction consequence, reduction likelihood
Concerning the order of derivation they write: "The metamodel is divided into five submodels that support different stages of a risk assessment. A risk assessment always starts with identifying the context of the assessment. A strengths, weaknesses, opportunities, and threats (SWOT) analysis may be part of this. After the context has been established, the remainder of a risk assessment can be divided into identification and documentation of unwanted incidents, risks, and treatments." These concepts are modelled in UML like this: class diagram for assets and stakeholders, use case diagram for threats, unwanted incidents, swot and treatment, risk as class. Their report also contains an extensive treatment of metrics and discussion of different types. We do not use much of UML, only use case diagrams to show use cases and misuse cases.
Moore, Ellison and Linger [MEL01] describe attack patterns by the attributes attack goal, preconditions (e.g. vulnerability), steps for carrying out the attack and postconditions (e.g. damage). As pre- and postconditions are part of a use case, we adopt this notation.

The concepts of assets, vulnerability and threats are implicitly used everywhere in the area of security assessment, see for example [CC99] and [BSI04], but the concepts are not clearly defined and even mixed because of lack of differentiation. Therefore, from the security and

reliability literature, we adopt only some single elements or ideas which were useful for our method.
The SQUARE project [Xie04] developed a framework for "Cost/Benefit Analysis for Information Security Improvement Projects in Small Companies" to support decision-making. To avoid an over-detailed risk modelling, they do not look at single risks and misuse cases, but treat seven categories of threat (Denial of Service, System Penetration, Sabotage of Data, Theft of Proprietary Info, Unauthorized Access by Insiders, Virus, Active Wiretapping). Each category contains several misuse cases, and each misuse case several incidents. Especially in small companies, the human resources and the data necessary for a more detailed analysis are not given. Therefore, where not available otherwise, they use financial and probabilistic data fram annual national surveys for each category of threats. They calculate and compare the risks, benefits and implementation costs of different combinations of realized preventions and optimize them. The challenge is to maximize the system value within real-life budget constraints. We will use their formulas for the prioritization of requirements (which will be done in working paper 2).

An overview about taxonomies of attacks is given by Killourhy, Maxion and Tan [KMT04]. They themselves develop a defense-centric taxonomy for anomaly-based detection, while the other taxonomies mentioned are attack-centric. They derive the four attack pattern classes "foreign symbol", "minimal foreign sequence", "dormant sequence", "non-anomalous sequence", what is interesting for the development of intrusion detection systems, but will not considered by us. Aslam [Asl95] for example distinguishs the three high-level classes "coding faults introduced during software development", "operational faults with which result from improper software installation" and "environment faults when a program is used in an environment for which it was not intended". We include these categories into our lists of vulnerabilities and threats. Others classify attacks according to their complexity of the signature, based on the intended effect (=what we call threat), the technique or means used, or the level of privilege required by the attacker. Not all of them fit into what we intend to do, but it is good to keep them in mind.

During the architectural design, for decisions and optimization some authors use the NFR as decision criteria or optimization goals.
Within the QOC notation [MYBM91], one can equate Question = design problem, Options = FR, Criteria = NFR.
Bob Blakley, Craig Heath, and members of The Open Group Security Forum published a catalogue of security patterns and a generic method using them to design a system architecture starting from given functionalities and a first architecture proposed by a stepwise optimization with regard to availability and security requirements [BH04]. They use patterns which describe the assets, threats and vulnerabilities of the system, but in the terms of "motivation", "applicability" and "consequences". They distinguish between resources and actors. The main difference between their risk analysis and ours is that they analyse a given first draft of a system architecture, while we do it on the basis of the requirements.
During the EMPRESS project, quality models have been constructed (see [DKVP03] and [DPB+04]), to link requirements to architecture and to verification and validation. More concretely, these quality models link quality attributes to means for satisfying them and to metrics for measuring quality, all in a tree structure. These quality models are intended to help mastering requirements and system change. Means are principles, techniques, or mechanisms that facilitate the achievement of certain qualities. Means are described by scenarios, which consist of stimulus and response, and a metric. Patterns are used to document architectural options. These lists of means and patterns will be cited by us.

ATAM (=Architecture Tradeoff Analysis Method) evaluates several architectural styles or solutions against each other, using quality requirements as evaluation criteria. In [KKB+98], Kazman et al. derive requirements and constraints on scenarios from non-functional requirements. In [KKC00], they use a terminology of sensitivity points, risks, stimuli, scenarios and responses, which looks like being symmetric to assets, threats, goals, misuse cases and countermeasures, although they are used in a different context. For our work in this paper, we want to keep in mind that Kazman et al. [KKC00] distinguish between customer, maintainer and developer scenarios, but also between use case scenarios (these involve typical uses of the existing system and are used for information elicitation), growth scenarios (these cover anticipated changes to the system), and exploratory scenarios (these cover extreme changes that are expected to "stress" the system). These types of scenarios are used to probe a system from different angles, optimizing the chances of surfacing architectural decisions at risk. The misuse case idea so far only applies to exploratory scenarios, while the misuser is neither customer nor maintainer or developer. We will consider them all, to be complete. For instance, some of the quality attributes only make sense when being considered in reference to maintainer scenarios (like maintainability).

From some more literature sources we use merely their extensive lists of threats, vulnerabilities and countermeasures, but did not use their concepts or methodology ([BSI89], [BSI91], [LBMC94], [Ran97], [CC99], [CNYM00], [AER02], [Ric03], [BSI04]).
For example, Chung et al. [CNYM00] decompose quality attributes (called "softgoals") in their aspects and derive means. These means we integrate into our lists of countermeasures. They also prioritize, document relationships and perform tradeoffs between conflicting quality attributes. They only loosely link non-functional requirements to functional requirements. For the moment we do no such conflict solving among requirements, but for future enhancements of our method (see working paper 2), we will certainly return to it.

The terms used by different authors differ a lot. Not only various terms are used for the same concept, but the same word has different meanings, and different concepts are mixed. Therefore, we edited table 1 which compares the terminologies used in five different literature sources. It also shows where we took our terms from.

In this work	EMPRESS Quality Model [DKVP03],[DPB+04]	ATAM [KKC00]	Misuse Cases [Fir03b], [MHN04]	Template of Sutcliffe & Minocha[SM98]	UML enhancement [OMG04]
Asset	---	(Tradeoff point)	Asset	---	Asset = target of evaluation
QA	Quality Attribute	Quality Attribute (Response)	---	NFR	QoS Category
Vulnerability	---	Sensitivity Point, (Architectural) Risk	---	---	Vulnerability
Threat	---	---	Threat	Expected failure	Unwanted incident
Misuse	---	Scenario	Misuse Case	Scenario	Threat

Case				+ damage	
Misuser	Stakeholder (customer and developer)	Stakeholder (customer, maintainer and developer)	Misuser	Agent	Agent
Misuser Attribute	---	Stimulus	Goal	Motivation	---
Countermeasure	Means	Response	Use case	Countermeasure	Treat-ment

table 1: The terminologies used in five different sources

3 Generalized Misuse Case Concepts

We chose the misuse case approach as a basis for examining NFR. The original Misuse Case principle is this: Misuse Cases take the view of a misuser whose goal is to misuse the system. Misuse cases foresee his/ her behaviour. From these misuse cases new system requirements can be derived. So misuse cases help to complement the system specification considering exceptions and threats, not only the intended and successful use.

We adopt the general idea to identify misuses with respect to all quality attributes of ISO 9126 and thereof further requirements which prevent misuse, i.e. help to satisfy the NFR. Doing so, we found that a more general terminology is necessary, because misuse cases are tailored to security requirements. By stepping back, we saw that behind the misuse cases and several other methodologies for analyzing NFR or evaluating different architectures as described in section 2, there lies a more general principle: An asset is to be protected from a threat/ misuse, and for doing so, countermeasures are defined. Therefore we think it appropriate to adopt concepts from these other methods where they make sense. See figure 1 for an overview of our concepts and terminology. We will now look at the definitions of the single concepts and discuss the generalizations which we thought necessary.

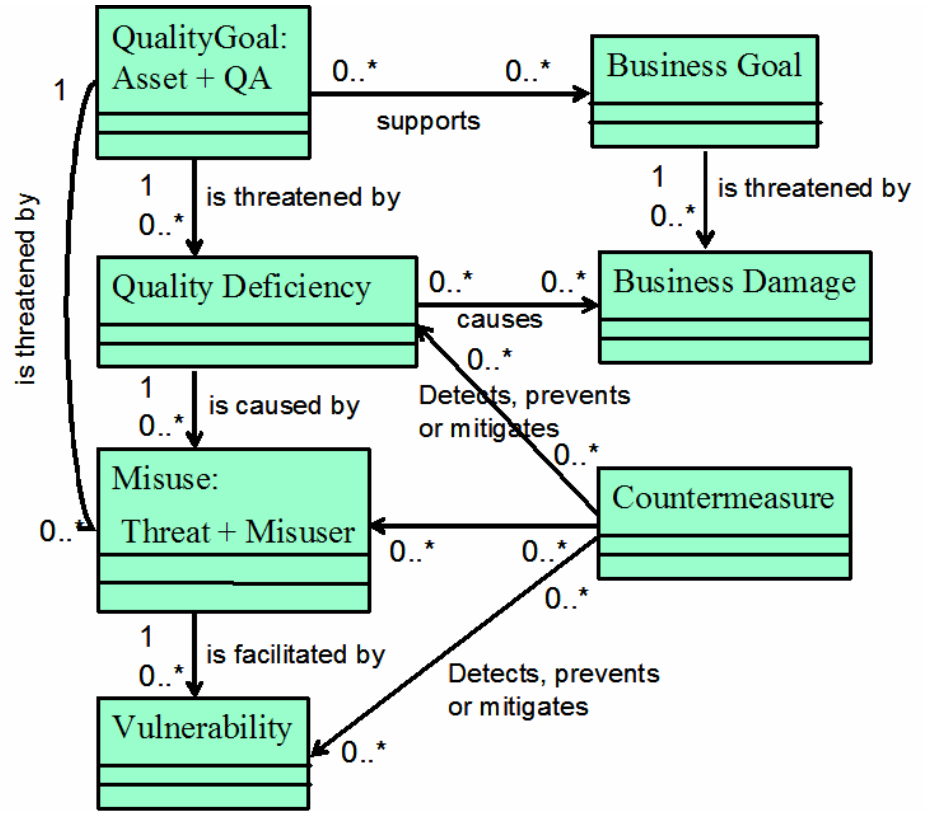

Fig. 1. The MOQARE concepts: see text for definitions and explanation

An *asset* is any part of the system to be protected. By "system" we include not only the software, hardware, network, but also the physical building, the company, the administrators, maintainers and users of the system. The misuse case literature expects "asset" to be "data, communications, services, hardware components, and personnel" [Fir03c]. ATAM [KKC00] mainly considers architecture components. For us, assets can be all of these (note: "services" can also be use cases). But: When data have to be secure, then not only the data, but also the quality attribute (QA) 'security' is protectable. Therefore, it is necessary to combine the asset with a specific *QA* or *NFR* to specify which quality of the asset needs protection. Thus, we define: It is the combination of an asset plus its NFR or a QA which is to be protected. We call this pair a *quality goal*. Usually one will start defining a QA like 'security' and during the discussion come to define a concrete security NFR for the asset. If possible, the NFR should include a metric to make it measurable, provable and testable.

The reason why the quality goal is to be protected is because it supports important *business goals*.

If the asset does not comply to the QA/ NFR, then we call this a *quality deficiency*. It is not necessarily the exact opposite of the QA. For example, if the quality goal is "availability of data", the quality deficiency can consist in temporary unaccessibility for all users or for certain users, irreversible destruction of the data, manipulation of the data or hardware, and many more.

The quality deficiency causes a corresponding *business damage* which threatens the business goal.

A *threat* is an action which would actively threaten the quality goal. It also is the cause of the quality deficiency. Firesmith lists the security threats "theft, vandalism, fraud, unauthorized disclosure, destruction, extortion, espionage, trespass" [Fir03c].

The threat is usually performed by a *misuser*, its driving force. This could be a person (hacker, users, administrators, etc.), other systems or forces of nature like fire and thunderstorm. In literature, the misuser often is described by a misuse goal or motivation. This might be the business damage (i.e. pure destruction), characteristic of the misuser (being disgruntled) or an

advantage for himself like theft leads to the thief owning the haul. As we do not want to mix goals, damages, etc., we either add the misuse motivation to the misuser description, like in "disgruntled employee" or in the misuse description (see below). 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 use (e.g. an online shopper "administrating" the account data of other clients, what is a use case for the system administrator), following a harmful goal, or a careless user violating security rules. All other ISO 9126 [ISO91] QAs, which we will regard here, are threatened by regular system users (i.e. end users) who try to use the system as intended, but fail for some reason. Not only end-users are relevant, but also administrators and maintainers (see ATAM [KKC00]), and developers. We also need this view because some of the QAs are only relevant to administrators, maintainers or developers. Most QAs refer to end-users, but recoverability and portability to administrators, maintainability to maintainers, time and resource efficiency, suitability and interoperability to end-user, developer, maintainer and administrator. To identify assets, misusers and threats, not only normal use cases (these involve typical uses of the existing system) are relevant, but also *growth scenarios* (these cover anticipated changes to the system; relevant for maintainability, interoperability, safety and portability) and *exploratory scenarios* (these cover extreme changes which are expected to "stress" the system; relevant for security, reliability, efficiency, recoverability). These three types of scenarios also were proposed by ATAM [KKC00].

The definition of the term 'threat' in literature is not clear. While Firesmith [Fir03c] by this term describes the anti-goal of a misuser, others [BSI04] mix misusers, forces of nature, reasons for misuse, vulnerabilities or the consequences of misuse in the same list.

Often, the threat is facilitated, made possible or even provoked by a *vulnerability*. A vulnerability is a property of the system and might be a code flaw or a design flaw or a flaw in the software development process, in operation or management, but also any – even wanted – property of the system, if it can be misused with respect to the quality goal. For example the system might provoke user impatience by its bad understandability or only because the user is used to another type of interfaces. On the other hand, not each system flaw needs to be a vulnerability. If there is no potential misuser for it, then the flaw is no vulnerability. So a system flaw or any system property must be evaluated against the quality goal to be protected, to decide whether there is a potential misuser who might try to threaten the asset´s QA. E.g., an open window is not necessarily a vulnerability. With respect to the quality goal "security of the money on the kitchen table" it only then is a vulnerability, if there is a potential intruder, i.e. the window is accessible to burglars. Maybe the window is on the 35th floor or is protected by strong iron bars. On the other hand, the closed window could be a vulnerability with respect to the QA "fresh air" or "no mildew". Equally, all properties of a system can be a vulnerability in one respect or the other.

A *misuse* describes the whole misuse scenario, including misuser, vulnerability, threat and its consequences (quality deficiency and business damage). The misuse is documented in the form and granularity of a misuse case which is similar to a use case. This means they are more elaborated than threats, which usually are described by a few key words. Misuses can be expressed as separate misuse cases, but also as an exception scenario being part of a use case. In the misuse case, the misuser is the actor, the vulnerability a pre-condition, the steps of the threat are described by a scenario, and the quality deficiency and business damage are the post-condition of the misuse case. Calling the vulnerability a pre-condition, we assume that the vulnerability is a necessary (but not sufficient) condition for the threat.

If the asset is a certain data base, its protectable quality attribute its time efficiency, the misuser an impatient user who needs a report urgently, the vulnerability the inefficiency of the report, then the threat is the misuser calling the same time-consuming report several times in parallel and the misuse case the full description of this scenario.

To handle threats and misuses we need *countermeasures*. We adopt this term from Sutcliffe and Minocha [SM98] as the most general expression for anything one does against a threat. Countermeasures can either detect, prevent [SO01] or mitigate a misuse [Fir03b]. The countermeasures defined by the work we build upon ([CNYM00][DPB+04][KKC00][SO00][SM98]), are new use cases, new or extended exception scenarios of use cases, use case NFR (including metrics for detecting a misuse), architectural or other constraints. Countermeasures protect the asset, but they depend on the special threat and vulnerability. Countermeasures can counteract against the threat, against the system vulnerability or against the misuse having the predicted consequences. They might reduce or eliminate the risk, i.e. the probabilities or the damage severity.
In the misuse case literature, one finds eight possible relationships between misuse cases and use cases (there being meant to represent the assets and the countermeasures). We interpret "include", "has exception" [FC99] and "extends" [SO01] as the relation between a use case and one of its exception scenarios, "threatens" [Fir03b] is the relation of a misuse to an asset, while "detects", "prevents" [SO01] and "mitigates" [Fir03b] leads from a misuse to a countermeasure, and "aggravates" and "conflicts with" [Ale02a] can mean a relationship among use cases, among misuse cases and between these two. These relationships can help to derive countermeasures from misuses, but also vice versa.
For prioritizing requirements throughout the concept hierarchy and to prepare trade-offs between conflicting requirements, we attribute a so-called "*relevance*" value to all requirements. This starts with the business goals (they can be rated in a currency, but also with 0 to 3 points), an evaluation of the threatening business damage, etc. The relevance of the business damage is the relevance of the threatened business goal times the percentage at which the damage destroys the business goal. The quality deficiency leading to this business damage has the same relevance as the business damage, but lowered if it does not always lead to the damage.
To describe the relevance of a misuse, we assess its risk: the probability of occurrence of the misuser, vulnerability and threat are defined, as well as the severity of the business damage, and the probability that this misuse leads to this business damage. As usually exact numbers or empirical data are not known, relative probabilities can be used (as does Firesmith [Fir03c], there called "vulnerabilities") and a simple classification of the business damage, e.g. with 1, 2 or 3 points. Damage can be loss of money, but also of reputation, client and end-user trust.
We build on the common risk formula which is *risk = probability of the threat times probability of vulnerability times damage* (see for example [ISO02]). We chose "*relevance*" as a more general term than "risk", as we want to use it also to characterize/ prioritize business damages and countermeasures.
We more generally say that for producing a misuse, we need the vulnerability (A) and the misuser (B) as pre-conditions. These might lead to the threat C (defined as an action) to take place, and this eventually leads to a certain damage D. According to the rules of probability calculation and assuming that A and B both are necessary but not sufficient pre-conditions of C, this leads to a conditional probability of the damage to be caused

$$p(D) = p(A \cap B) \cdot p_{A \cap B}(C) \cdot p_{A \cap B \cap C}(D)$$

If the probabilities of A and B are independent, then $p(A \cap B) = p(A) \cdot p(B)$ with $p(A)$ = existence of vulnerability and $p(B)$ = existence of misuer with given characteristics, (as sometimes not simply a user, but an impatient user is the actor of a misuse). The conditional probability $p_{A \cap B}(C)$ denotes the probability that the threat is performed if both the vulnerability is given and the misuser exists. $p_{A \cap B \cap C}(D)$ describes the probability that misuser and vulnerability given, the threat happening, that expected business damage D is caused. As has been done in the risk formula, we multiply this probability $p(D)$ with the *severity* $s(D)$ of

the damage to calculate the *relevance of a misuse* consisting of vulnerability A, misuser B, threat C and business damage D:

$Relevance = p(A \cap B) \cdot p_{A \cap B}(C) \cdot p_{A \cap B \cap C}(D) \cdot s(D)$

For prioritizing requirements, the *relevance of a countermeasure* will also play a role. We define the relevance of a countermeasure to be equal to the relevance of the misuse if the countermeasure does prevent the misuse totally. Otherwise it is equal to the difference between the relevance of the misuse without the countermeasure and the value with the countermeasure working. I.e. if the countermeasure reduces the probability of the misuse by 30%, then the relevance of the countermeasure is the relevance of the misuse multiplied by 30%.

4 The MOQARE Method

For now, we have not given any method or direction on how to derive these concepts described in section 3, but only showed how they relate to each other. This section proposes a method which finally has worked well in our case study (see section 6). Its purpose is to define new requirements like FR, NFR, constraints and further requirements supporting or assuring that the asset has the protectable QA, by considering misuses.
Our concepts could also be used to decide between given architectural alternatives by evaluating them against the quality goals, but this is not the focus here.

MOQARE starts with the functional requirements (i.e. functional description) of a planned or existing system. The requirements engineer is guided by a four steps process and supported by checklists. In this section, we outline the procedure of the method, the checklists are given in the annex. The procedure identifies the concepts in the following order, which is not to be understood as an obligatory order but rather a guideline. As requirements elicitation is a creative activity, the steps can also be performed iteratively, and they even must be repeated if a countermeasure is a new quality goal.

1. find the quality goals (based on business goals, quality deficiencies, and business damages)
2. describe misuses (based on threat, misuser, vulnerability)
3. define countermeasures
4. if necessary, re-start the cycle

1.) A quality goal (= an asset and QA) is valuable because it supports a *business goal*. Therefore, it makes sense to start with the definition of the business goals. What is the essential of your business? This might be a business process or a business goal like “30% of market share”. According to Regev and Wegmann [RW05], there are the following types of goals:

- achievement goal: “Achievement goals are objectives of an enterprise or system” and: “An achievement goal is satisfied when the target condition is attained.”
- Maintenance goal: property that holds in current and all future states
- Avoidance goal: specifies a state that is to be avoided
- Softgoal: “a condition or state of affairs in the world that the actor would like to achieve, but unlike in the concept of (hard) goal, there are no clear-cut criteria for whether the condition is achieved, and it is up to subjective

judgement and interpretation of the developer to judge whether a particular state of affairs in fact achieves sufficiently the stated softgoal“ [ITU01]
    - Belief: “Beliefs are used to represent design rationale. Beliefs make it possible for domain characteristics to be considered and properly reflected into decision making process, hence facilitating later review, justification and change of the system, as well as enhancing traceability” [ITU01]

Dardenne, van Lamsweerde and Fickas [DVF93] define the following goal patterns:
- Achieve: a property which holds in current or some future state
- Cease: a property which holds not in current or some future state
- Maintain: a property which holds in current and some future states
- Avoid: a property which holds neither in current nor in some future states
- Optimize: maximize or minimize an objective function

The following list gives a summary of techniques for identifying stakeholders´ goals [RW05]:
- Understanding stakeholders´ problems and negating them
- Extracting intentional statements from:
    - Interview transcripts
    - Enterprise policies
    - Enterprise mission statements
    - Enterprise goals
    - Workflow diagrams
    - Scenarios written with stakeholders
- Asking “how” and “why” questions about these initially identified goals in order to go up and down the goal hierarchy
- Asking “How else” questions to identify alternative goals
- Searching for action words that describe a state that is to be achieved, maintained, avoided, etc. (keywords for achievement goals: achieve, make, improve, speedup, increase, satisfy, complete, allocate; keywords for maintenance goals: maintain, keep, ensure, avoid, know, monitor, track, provide, supply, found out)
- Asking what goal a given statement exemplifies and what goals are blocked or obstracted by a statement
- Asking why an identified goal is to be achieved or maintained
- Looking for statements that guide design decisions at different levels of the IT system or enterprise
- Considering pre and post conditions of already identified goals
- Using domain knowledge
- Identifying goal obstacles and constraints
- Considering possible scenarios for goal achievement and obstruction

After having identified the major business goals, think of what *quality deficiency* might threaten them and cause which *quality deficiency* and *business damage*. For doing so, we use the ISO 9126 [ISO91] hierarchy of quality attributes as a checklist (see annex A.3). We use the two levels of this standard and include a third level for security as described in the annex. A quality deficiency can be the complete or the partial or temporary lack of the quality attribute. Probably all quality attributes must be satisfied to a certain degree, but which of them do you want to study more closely, which are essential for the business, which quality want you to be designed into the system or being protected to an especially high degree?
Each quality deficiency will lead to a certain *business damage*. Prioritise these business damages according to their *severity* (for example by attributing 1, 2 or 3 points), where the

high value must be attributed to the high damages. This weight factor will be used later-on for prioritization of NFR and for trade-offs in case of conflicting requirements.
To deduce the *quality attribute* to be protected from the quality deficiency is straightforward. Now for each quality attribute derive the affected *asset*s. An asset can be domain data, roles, system/ hardware/ software components, tasks, activities, use cases or services, communications (computer network or interfaces). The granularity of assets is different in various literature sources. One might name "data base" but also factorize further and name the use case "archiving" or the system component "user interface" or "database table 'credit card numbers'". As a checklist, you can use the list of assets in annex A.2 or the hierarchy of assets given in A.5. Of course, it is far from being exhaustive, but it helps to think about the specific assets in your system.
The result of this high-level threat analysis will be the quality goals (= asset + QA) to be protected, like "confidentiality of credit card number" or "availability of web shop" or "time efficiency of a use case Y".
As can be seen in figure 1, the relationships among the concepts are complex and can not easily be depicted in the form of a tree. It as well makes sense to present, for each business goal, the business damages and quality deficiencies which threaten it (as has been done above) or, vice versa, think about which quality goals support the business goal. Both ways will lead to the quality goals which describe high-level quality requirements for the system.

2.) Describe *misuses* in the form of a misuse case: A misuse case is described by an action (threat), the actor (misuser), a vulnerability (pre-condition) and the quality deficiency or business damage caused (post-condition).
We consider the context of user use cases (these involve typical uses of the existing system and are used for information elicitation), growth use cases (these cover anticipated changes to the system), and exploratory use cases (these cover extreme changes that are expected to "stress" the system), as was proposed by ATAM [KKC00]. For our application, this also makes sense, as maintainability and portability refer to growth use cases, security to exploratory use and usability to normal use.
As the same threat can be performed by several different misusers with a different course of event (e.g. overload produced by a hacker performing a denial-of-service attack or an impatient user starting the same request several times), we start with the identification of the *threats* and might then derive one misuse case per misuser. In the literature, lots of potential threats are known. We expect them to be reusable. Therefore, we have gathered many of them in checklists. For each quality attribute, there is a separate threat list (see annex A.4). Theft for example threatens availability and confidentiality, but never usability or efficiency.
These lists also contain proposals for the potential *misusers* and countermeasures. For some threats, there are several versions like the "intentional data corruption by intruders" and the "unintentional data corruption by user". For your system, you probably can express more clearly who the misuser can be, like "accountant". Think of criminals as well as normal users, maintainers and administrators. Think of destructive goals and ignorance. Don´t forget forces of nature, other systems, or the system environment (technical, social, political, etc.). For practical work, you probably won´t want to identify all potential misusers, but only those who are most relevant, i.e. those with the highest probability or who cause the most harmful damage.
The threats are facilitated by *vulnerabilities*, or the misuser exploits a vulnerability. For a given asset, like a data base or a certain operating system, many potential vulnerabilities are known. Therefore it should be possible to compile reusable lists of vulnerabilities and their countermeasures. They can be used as checklists for not forgetting the best-known vulnerabilities and to get ideas about further vulnerabilities. Here, you will also concentrate on the most relevant ones. In annex A.5, you find such lists of known vulnerabilities in

different assets. We believe that these lists need not be compiled for each quality attribute separately, but depend mainly on the asset, as the same property/ vulnerability can be misused with respect to different QAs.
In the post-condition of the misuse case, document the *quality deficiency* or *business damage* caused by the misuse. They are also given in the threats checklists.
For later trade-off among contradicting requirements, the relevance of a misuse is calculated according to the formula
$Relevance = p(A \cap B) \cdot p_{A \cap B}(C) \cdot p_{A \cap B \cap C}(D) \cdot s(D)$
as was described in the preceding section.

3.) Define countermeasures:
For each misuse case, try to find countermeasures against the threat, the misuser, the misuser´s motivation, the vulnerability and against the business damage. Countermeasures can either detect, prevent or mitigate. They can be use cases, new or extended exception scenarios of use cases, use case NFR (including metrics for detecting a misuse), services, architectural constraints, user interface constraints, constraints on project/ software development, constraints on maintenance, or another quality goal. Often, it makes sense to add a metric to the countermeasure.
Our lists of threats and vulnerabilities also provide countermeasures (annex A.4 and A.5), but we do not claim them to be complete. Moreover, they are quite general, while the ideal countermeasure is concrete, realisable and often also system-specific.

4.) if necessary, re-start the cycle at step 2
A countermeasure can also be a new quality goal. For example the usability of the user interface helps to improve the integrity of the data entered by the users manually. In this case, the elicitation of NFR is not finished when finding all countermeasures but must start anew with the newly defined quality goals.

This method leads to hierarchical results which can be presented in the form of a tree, a "misuse tree", similar to attack trees [LBDJ03] and quality models [DKVP03][DPB+04] , but with different concepts. Such a tree presentation makes sense because for each business goal, there are several business damages, quality deficiencies and quality goals, for each quality goal several threats and for each threat several countermeasures.
A misuse tree has the following levels, from top to bottom:

- business goal
- quality deficiency
- business damage
- quality goal
- misuse (including threat, misuser, vulnerability, quality deficiency, business damage and relevance)
- countermeasure
- quality goal
- misuse
- countermeasure
- …

An example of such a misuse tree can be seen in the case study.
The derivation of the assets and QA could be seen as a first cycle of threat analysis on the business level, which bridges the gap to the requirements. On the requirements level, the analysis of threats often leads to countermeasures which are quality goals themselves, so a

new cycle begins. We do not know by now, how many such levels or cycles one can find by a thorough investigation of a complex system, but the number of combinations of asset + quality goal is finite (= “number of possible assets” times “number of quality attributes”), and somewhere there will be loops, when quality goals are repeated or at least countermeasures re-appear.

This analysis can be performed in two alternative ways: If the goal is acomplete analysis of all potential threats (actual, future, already prevented, unprobable), then all potential threats are considered, and the probability and damage estimated later-on. But in practical work, one often wants to analyse the actual quality of the system (or system draft) and the potential ways of efficient improvement. Then, a threat or a vulnerability is not relevant to be considered, if there is already an effective countermeasure foreseen against it. In the case study, we concentrated on relevant misuses, i.e. with significant probability and damage.

5 Integration of MOQARE with TRAIN

The software engineering group at the University of Heidelberg, for software requirements engineering uses a methodology called TRAIN which is supported by a tool named Sysiphus (see: http://sysiphus.informatik.tu-muenchen.de/). TRAIN not only concerns the requirements but also design and testing of software, i.e. the whole software lifecycle. But here we refer to the requirements part of TRAIN only.
In section 5.1, we give a description of the TRAIN elements describing FR. Then, in section 5.2, we describe how NFR are captured within this framework and how they are integrated with the FR, and especially how our misuse case concepts fit here.
We expect that MOQARE can also be integrated into other methods for elicitation of FR, as has been done for the security analysis only by Breu (Integration of security analysis into the use case based PROSECO) [BBH03][Bre05] and Meyer, Rifault and Dubois (integration of risk analysis into i*) [MRD05].

5.1 Functional Requirements in TRAIN

TRAIN describes requirements on four levels: Task Level, Domain Level, Interaction Level and System Level (see figure 2).

Case study in section 6 was removed because it is confidential

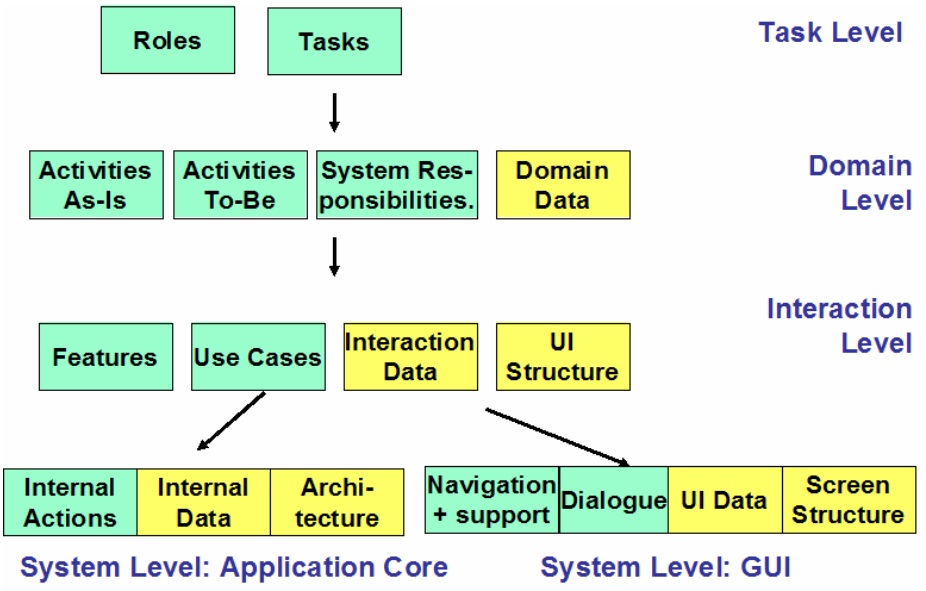

figure 2: levels of TRAIN

Task Level
On the Task Level, decisions about **Tasks** and **Roles/ Actors** are taken (Actors are persons and systems which take part in the supported processes), without defining here which tasks later-on will be performed by the system and which one by the user. The role and task descriptions represent the FR on task level.

Domain Level
On the Domain level, the tasks defined on Task Level are detailled in smaller steps (**Activities**). The activities/ business processes as they are right now (“**as-is”**) as well as the activities “**to be”** are defined. Only the tasks determined on the Task Level are considered here. On this level, it is decided how the business process will change by use of the IT system. Those activities to be supported by the system are identified (**System Responsibilities**) as well as the data to be managed by the system (**Domain Data**).

Interaction Level
The Interaction Level focuses on the human-machine-interface. Here is decided how the user interacts with the system. Therefore, the structure of the workspaces is defined (**UI structure:** In which context is the user allowed to call which functions and data) and how (e.g. in which order) the user performs her/ his activities with the system (**Use Cases**). For allowing this, the system is to provide **Services** (System Functions) which are also described here. The **data model** is refined here, describing how the data exchanged and manipulated between and by user and system are related to each other and on which user interfaces they are to be accessible.
One use case can contain several scenarios: the main success scenario and alternative scenarios for describing variations and exceptions. A **Scenario** describes one potential specific event flow of a use case. It is an instance of a use case. Vice versa, a use case is the abstract description of a finite set of scenarios.
Services document which data are the input, which are manipulated by the operation, how the result is to be calculated, which exceptions are to be expected, etc.. On the interaction level, they are still described from the user viewpoint and do not represent a technical specification.

System Level
On the System Level, a blueprint for a technical realisation is prepared. The aim of this level is to allow a structured, flexible realisation of the specification in a specific technology (e.g. object orientation). Here, decisions are taken about the graphical user interface (**GUI**) and the **Application Core**. The GUI part contains considerations concerning the **user interface structure** and **navigation**, while in the area of the application core, the transition from requirements specification to object oriented analysis is performed. The result of this process is an analysis class diagram, which serves as the basis for the detailed design. For this, in the TRAIN process a method by Ivar Jacobsen (OOSE) is applied, which uses the artefacts already realized as input, especially the domain data diagram respectively refined data model, the use cases and system functions, for developing the so-called analysis class diagram.

5.2 Integrated Method

TRAIN allows the documentation of FR and NFR, and MOQARE derives as well NFR as FR. How can the two methods and their results be integrated? As will be described in more detail in our case study, we started with the **business goals and** a description of the **FR**. This functional description and business goals are the basis of the Misuse analysis, as otherwise no assets or misuses can be defined. If you do not know what is to be protected, you do not know what could potentially happen. The functional description of the system need not be complete. A description of tasks and domain data would be sufficient, but the more detailed the functional requirements are known, the more detailed the misuse analysis can be performed also.
Then, MOQARE followed and starting from the business goals derived business damages, quality deficiencies, quality goals, misuses and countermeasures. The results of this analysis can be and should be presented in two presentations: firstly, there will be the misuse tree which shows the logical relationships of the concepts of MOQARE. As the countermeasures derived can be of different types (e.g. new functional requirements, constraints on FR etc.) they should also be arranged according to their type. This can be done by integrating them into the presentation of the FR, here the TRAIN levels.
When a countermeasure is a new quality goal, then a new iteration of MOQARE starts to find the misuses threatening this new quality goal.
If a countermeasure is a new FR, then a new TRAIN cycle might be started. Is the new FR a task, for instance, then it will be detailed into new activities, use cases, maybe add domain data, etc.. These new requirements can lead to new misuses which now must be included into the misuse tree.
This means that not only all consequences of new requirements must be considered, but also a regular review of the requirements with respect to completeness will make sense.

5.3 Non-functional Requirements in TRAIN

On all four levels of the TRAIN process, NFR respectively constraints can be captured, for example task constraints or use case constraints. To allow an extensive and systematic treatment of NFR, we now integrated the MOQARE concepts into the TRAIN levels, as is shown in figure 3.
This makes sense because the analysis of the FR as well as MOQARE lead to use cases or constraints, and if a sensible trade-off of requirements is to be done, this is only possible based on an integrated presentation of all requirements.

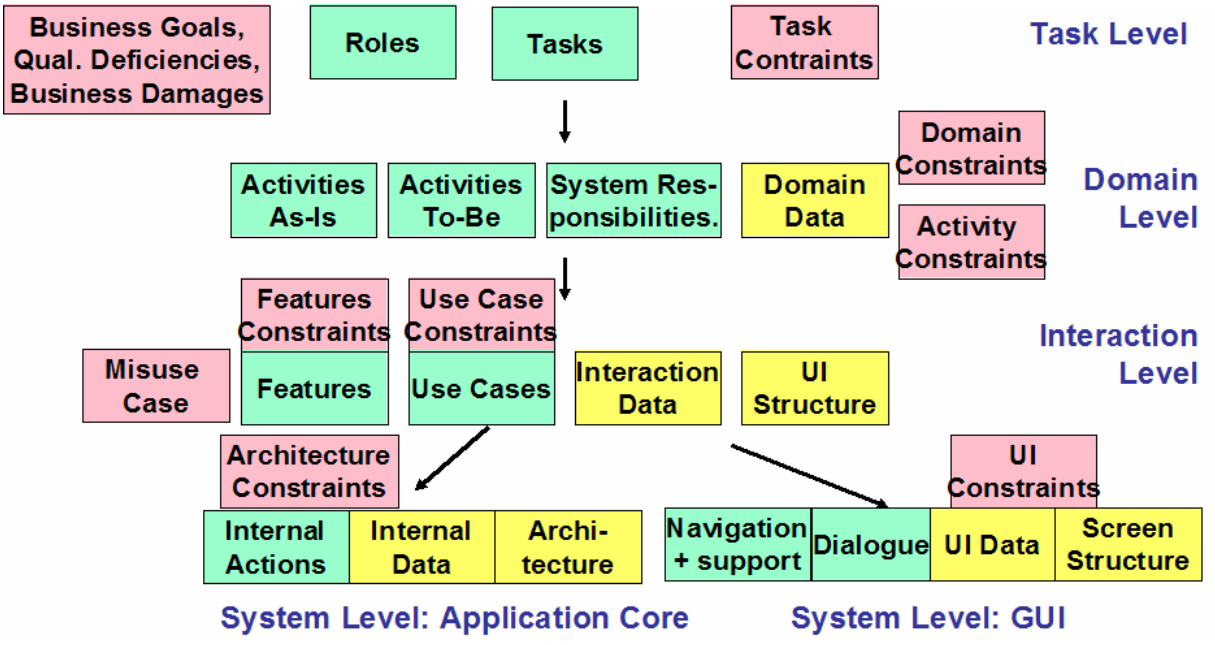

figure 3: levels of TRAIN, including the Misuse Cases and Quality Constraints

On all levels

A concept which can not be attributed to any specific level of TRAIN, is the **quality goal (asset + QA)**, because assets can be found on all levels. An asset is no new concept, but existing concepts are chosen to be important enough to be called an asset. Assets can be roles or tasks (domain level), data, activities (domain level), use cases and user interfaces (interaction level) or services, hardware or software components and their communication (system level).

Countermeasures can also be found on all levels because they can be: tasks or activities, quality constraints on tasks or activities, use cases or services, new or extended exception scenarios of use cases, use case or service constraints (including metrics for detecting a misuse), architectural or other constraints like Constraints on User Interface.

Task Level

On this level, which is the business level, we see the right place for the **business goal** and the **business damage**. The tasks can be constrained by "**Quality Constraints of User Tasks**", which might result from the threat analysis as a countermeasure or/ and be part of a quality goal where the task is the asset and the "Quality Constraints of User Task" the quality attribute. These two alternatives exist for all other constraints described below.

Domain Level

On the Domain Level, one can define **Domain Constraints,** which are constraints typical to the application domain respectively environment of the system, like the time pressure on the system users. These domain constraints often show up to serve as vulnerabilities within the misuse analysis. Domain Constraints can also be neutral information like the number of

system users, or other influences from outside like political, market related, standards, technical strategies, cultural, organisational or physical constraints.
Similar to tasks, Activities also can be associated with "**Quality Constraints on Activities**", which also can express a countermeasure or/ and be part of a quality goal.

Interaction Level
Use Cases as well as Services can have their "Quality Constraints on Use Cases" and "Quality Constraints on Services" respectively.
The **Misuse Case** finds its place on this level. It has a similar form as the Use Case. The main difference between these two is that the Use Case describes the wanted behaviour and the Misuse Case what must not happen. Therefore, it is tied to one or several countermeasures.
There are two alternatives concerning the representation of Misuse Cases. They can be represented as a separate Misuse Case, but also can be described by an exception scenario of a Use Case. This is because many misuses happen during normal use. Even if a normal user tries to manipulate her collegue´s account data without permission, we could consider this to be a special scenario within the use case "manipulate your own account data". We decided to model misuses always by separate Misuse Cases because for a complete misuse description we need not only its steps but also vulnerabilities (pre-condition), damage/ quality deficiency (post-condition) and links to the countermeasures. As the use case describing the normal use will have its own pre- and post-conditions, there is danger to mix up normal and unwanted use. Therefore, we rather advice to link misuse cases to the use cases which they represent an exception of.

System Level
On this level, "**Architectural Constraints**" and "**User Interface Constraints**" are contained, which frequently are countermeasures.

6 Case Study

The method(s) described above were applied to one software system, the Uveitis database. In section 6.1 only the FR are described, on the task, domain and interaction level. In 6.2 the MOQARE is applied to define the NFR, on the basis of the functional description of section 6.1. This leads to a "misuse tree" which shows the causal dependencies between the analysis results. In section 6.3, examples for the integrated results are shown.

[…]

6.1 Discussion of the results and the course of the case study

What do we learn from this analysis? We can compare our results with former experiences with the same case study where we did an unstructured analysis or used former versions of the method described here.
The method supports a systematic investigation of non-functional requirements, based on and integrated with the functional requirements. It was well guided by the four steps of concept elicitation and by the checklists. The idea of the method could easily be understood by the stakeholders. Problematic was the estimation of probabilities and costs, as they were not easy to estimate and a lot of numbers were necessary. This made the procedure hard from the moment on when the relevance values were to be estimated. A support by standard values (e.g. statistical averages) would be nice, but hardly be feasible as the conditions of the specific environment must be considered.

Although the resulting misuse tree looks complex and extensive, it is much clearer and more compact than our first unstructured approaches. Therefore, we believe that it is the most simple representation of the complex results of a complete NFR elicitation.
In an older version of the method, we only analysed the system, without relation to the business goals. This led to an unstructured list of quality goals, which was probably not complete, but its completeness could not be judged. The list also included redundancies like when a use case must be usable, then the user interface used in this use case also must be usable. When finally starting with the business goals, we first find quality goals which directly support the business goals, and then – on lower levels of the misuse tree – such which support them indirectly. This does not only lead to a clear prioritization among quality goals, but also helps to find all of them, because they all must be related to the business goals in some way and appear in the misuse tree. The prioritization was quantified by calculating the "relevance" of quality goals, misuses and countermeasures. No such redundancies as mentioned above were observed.
In the misuse tree, you find the different quality attributes (QA) and see how they are all related to each other. For example data integrity depends both on security and on usability, and usability depends on time-efficiency. Therefore it makes more sense to regard all quality attributes in an integrated approach, not ignoring the special knowledge from the HCI community concerning usability or from the security community, but integrating them.
It made sense to present the results of the analysis in a tree as each quality goal is threatened by several threats and each threat has several countermeasures. But sometimes loops did appear. For example, the authorization concept is a countermeasure against two threats (both threatening the quality goal "data + integrity"). And secondly, the integrity of the data is threatened by the threat "unintentional corruption of data". As one of the vulnerabilities leading to this threat would be the non-availability of the data or software, one of the countermeasures is the quality goal "data + availability". But this goal again is threatened by "lack of data integrity" and protected by the quality goal "data + integrity". Here, we enter into a loop.
As we decided to do an iterative requirements elicitation, at each iteration former versions of the misuse tree could be used as an interview guide for the next iteration which consisted in a review of the former results and then a branch was chosen where the interview was to continue to bring forth new results. Also, approximate estimations about how much of the results were still missing, could be done by counting the number of quality goals still being without threats and countermeasures (knowing that the leaves of the tree always are countermeasures and that further quality goals can arise as countermeasures).
As the aim of the analysis is to find system specific, realizable requirements, the checklists provided by us are a good support for creativity, but the domain-specific wording should always be preferred. During the analysis, we often reached points in the tree where quality goals were defined which could be satisfied by known solutions which do not depend on the special software investigated. Take for example the intrusion of hackers. As countermeasures, one can propose intrusion detection and all measures which prevent intrusion or at least make it more difficult. Such solutions are known. Sometimes there exist products on the market, like intrusion detection software, or handbooks or specialists can tell a whole bundle of countermeasures to be taken to prevent intrusions. This was the point at which we stopped the analysis. The corresponding countermeasures can be taken from our general lists in the annex and the specialized literature concerning for example intrusion detection and prevention.
Some countermeasures at first looked trivial and common-sense like "compliance to known usability rules" or "good testing". But if our aim is a complete description of all requirements, then this is a good result and shows that this method helps to also derive the "tacit assumptions" so much seeked for by requirements engineers. We believe that the logic behind this observation is this: These trivial requirements are considered to be trivial, because they

prevent misuses which are relevant to most software systems. But nevertheless, they are important to protect a business value. Otherwise, they would not have appeard in our analysis.
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, but also because they are relevant to quality. Software quality is the result of good software development and good project management, 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 usually not included in a requirements analysis. But it now seems obvious to include them in the description of requirements to be complete, as data cleansing and maintenance in fact are tasks one wants to perform in the system, as well as the support of the functional requirements which support the business processes. Why? Because 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!
We started with 14 use cases, but the number of the misuse cases counts several dozens. We can not give the exact number. As we mentioned above, the analysis was stopped at those points where the discussion starts being too general and standard solutions are known. For example, one can think of many, many misuse cases for intrusion into the network, depending on the vulnerability which is misused.
In the case study, one soon could see that some misuses are clearly more relevant than others. The relevance values look like a good means to prioritize countermeasures. Sometimes, though, during the case study when comparing the values for different countermeasures, they did not seem appropriate. This is to be expected because they are calculated from several factors which are not easy to quantify. Therefore, it made sense to compare the relevances of different countermeasures and to compare their expected order. Then a re-estimation of the probability and relevance values was made.
Not considered here so far are the risks and adverse effects of countermeasures. This would be possible, but has been done for only one case, as can be seen in the lowest line of the misuse tree. For the configuration of the handheld, two alternatives are possible, each of which leads to another threat (either creating of doublets or loosing data input on the handheld). Such threats provoked by a countermeasure will be important for the trade-off of requirements, but is not important in this working paper. It will be treated later. It is the same for other relationships between countermeasures. For example, the “user training about security policy” only makes sense after “define security policy”, although it was attributed a higher relevance value, because the mere definition of a policy does not improve much by itself. Such questions are delayed to working paper 2.

7 Perspectives for Further Research

MOQARE has been used successfully in a case study for a software which already exists and has been used in pilot operation. We would also like to use it already in the phase of requirements elicitation or software design to see how well one can predict potential misuses with the aid of this method.
In the next step, we will have a closer look at conflict identification and negotiation, for conflicts among functional and non-functional requirements. How to find the requirements to be implemented? Not only the relevance and cost of countermeasures as defined above will be important, but also to consider threats provoked by countermeasures and dependencies among them like chronological order. We will also integrate the requirements analysis as described above with the following step in software engineering, the architecture design. The

requirements presentation developed in this working paper will be an important input for the trade-off between requirements and for the architectural design.
Furthermore, the generation of test cases out of requirements will be interesting, especially as the NFR must not be forgotten.

8 Summary of the Paper

This paper develops concepts and a method for a systematic analysis of NFR of a software system. This approach is based on the concept of misuse cases and on reusable lists of threats, vulnerabilities and countermeasures. This method can be used either separately when already knowing the FR, but also integrated in the method for elicitation of FR. Here, we integrated it to the TRAIN method.
The elicitation of non-functional as well as functional requirements is illustrated by applying them to a case study. We believe that an analysis of NFR in terms of quality goals, threats and countermeasures helps to complement software and project requirements. To support a complete view on system quality (in particular all QAs), we consider not only end-users, but also system administrators, maintainers and intruders to the system. We take account of use, growth and exploratory scenarios. An important value of our approach is its general applicability to all QAs and the integration of NFR and FR. We believe that the requirements analysis and documentation method described in this paper has been optimized with regard to understandability, clarity and completeness of results. By including a quantitative measure for the relevance of a misuse and countermeasure, we set the basis for requirements prioritization and trade-offs in case of conflicting requirements (which is not treated in this working paper, but in the next one).

Annex A Checklists

It was proposed by Firesmith [Fir03c] to compile reusable lists of threats and countermeasures, as similar threats are relevant to different systems, and many countermeasures are already known. As we have seen in section 4, such lists can well complement a systematic method for NFR assessment. Our lists presented here, are meant as checklists, i.e. suggestion and help, which can be used for applying to a concrete system. They are not meant to be complete, but they are a synopsis of the results of several reliable literature sources.
Below we summarize our checklists used in the method described in section 4.
This annex section contains the following lists:

1. business goals
2. assets
3. QAs
4. for each QA a list of threats and their countermeasures
5. for each asset a list of vulnerabilities and their countermeasures

Alternatively, we might have provided for reusable lists containing whole misuse patterns, for each pair of an asset plus QA the corresponding vulnerabilities and threats and their countermeasures. This would be easier to use. But we did not do it for several reasons, the two most important are: While there is a manageable number of QAs and assets, the number of the combinations is quite high. So, our lists would have taken much more space than they do anyhow. Secondly, we mainly wanted to summarize the knowledge of others, not write an exhaustive treatise about security, usability etc.. As our sources did not provide information which fit the pattern we use, it would have demanded well-founded expertise to bring them into such a form and fill in the gaps left open by the sources. This we did not want to do but leave this to the specialists. We mainly provide for the method, not the content. These lists are our attempt to categorize and re-use the knowledge of others.

A.1 Business Goals

For identifying the **business goals**, the following categorizations can help:

- Goals belong to the following **five dimensions** [Wie02]:
  - Product size
  - Quality
  - Staff
  - Cost
  - (calendar) time

- goals can belong to different **viewpoints**, e.g. according to the Balanced Scorecard [KN92]
  - Financial
  - Customer
  - internal processes
  - learning & growth

or to the Balanced IT Scorecard BITS [Iba98],[BAM01]
  - Financial: How do our software processes and SPIs add value to the company?
  - Customer: How do we know that our customers (internal and external) are delighted with our product?
  - Process: Are our software development processes performing at sufficiently high level to meet customer expectations?
  - People: Do our people have the necessary skills to perform their jobs and are they happy doing so?
  - Infrastructure & Innovation: Are process improvement, technology and organisational infrastructure issues being addressed with a view to implementing a sustainable improvement program?

- One can distinguish the following **goal patterns** [DVF93]:
  - Achieve
  - Cease
  - Maintain
  - Avoid
  - Optimize (maximize or minimize)

A.2 Assets

During our literature research, we found that all assets belonged to the groups "data, communications, services (might be use cases), hardware components, personnel" as defined by Firesmith [Fir03c].

Others know "seven categories of targets: The first three of these are "logical" entities (account, process or data), and the other four are "physical" entities (component, computer, network, or internetwork)." [HL98]

In the terms of the TRAIN concepts, our checklist reads like this:
- domain data
- roles
- system components
- hardware components
- software components
- tasks, activities, use cases or services
- communications (computer network or interfaces between systems)
- user interfaces

For practical use, more concrete assets will be necessary. Many examples will be named in section A.5.

A.3 List of QAs

Following the standard ISO 9126 [ISO91], we started with this hierarchy of QAs:

- Functionality (Security, Suitability, Accuracy, Interoperability, Compliance)
- Reliability (Maturity, Fault tolerance, Recoverability)
- Usability (Understandability, Learnability, Operability)
- Maintainability (Analysability, Changeability, Stability, Testability)
- Portability (Adaptability, Installability, Conformance, Replaceability)
- Efficiency (Time behaviour, Resource behaviour)

Security/ safety we factorize further into availability, integrity (completeness and accuracy), privacy/ confidentially, and operational security [CNYM00], and: immunity, survivability [Fir03c]. Reliability will be regarded in terms of Maturity and Fault tolerance, and separately in terms of Recoverability. The two aspects of Efficiency (Time behaviour, Resource behaviour) will also be treated separately.

DIN EN ISO 9241 [ISO92] in part 9241-10 lists seven criteria for usability:

- suitability
- understandability
- operability
- conformance to expectation
- Fault tolerance
- customizability
- learnability

We find them in ISO 9126 also (see above), but differently structured. This shows, that different classifications would make sense.

So, our hierarchy finally has the following three levels:

- Functionality
  - Security
    - Operational security
    - Availability
    - integrity (completeness and accuracy)
    - privacy/ confidentially
    - immunity
    - survivability
    - safety
  - Suitability
  - Accuracy
  - Interoperability
  - Compliance
- Reliability
  - Maturity
  - Fault tolerance
  - Recoverability
- Usability
  - Understandability
  - Learnability
  - Operability
- Maintainability
  - Analysability
  - Changeability
  - Stability

    - Testability
- Portability
    - Adaptability
    - Installability
    - Conformance
    - Replaceability
- Efficiency
    - Time behaviour
    - Resource behaviour

A.4 QAs with lists of threats and their countermeasures

QA: Availability

Threat -> consequence	**Misuser**	**countermeasure**
Theft -> permanent loss and non-availability of asset (hardware, service, etc.)	Thief, e.g. employee, external personnel, spy	physical protection (e.g. locked doors and windows, alarm system, monitoring, video surveillance), other security measures, building plans must not highlight assets, doorman, patrols, control and restrictive handling of physical access rights, appropriate key management, supervision or monitoring of external personnel
Vandalism, destruction -> permanent loss and non-availability of asset (hardware, service, etc.)	Hacker, disgruntled employee, cyber-terrorist, vandal	Like above
error, like unexpected input or switch-off of server during operation -> system breakdown -> temporary system unavailability	user, administrator, maintainer	Reliability and fault tolerance,, monitoring, training and documentation, control and restrictive handling of access rights to software, interdiction of use of software being not released internally
error like erroneous deleting of data, incorrect saving of input data -> Loss of data	User, administrator, maintainer	Usability, foresee potential user errors, control and restrictive handling of access rights to software, interdiction of use of software being not released

		internally
Damage of hardware by accident -> destruction of data, data base or hardware, data loss	thunderbolt, fire, water, cable fire, undue temperature or humidity, dust and dirt, technical catastrophe in the periphery, big event, storm	Physical protection against storm, thunderbolt, water, fire, dust and dirt, redundancy, air conditioning, compliance to fire prevention rules, hand fire extinguisher, safety doors and windows, room assignment regarding fire loads, Fire-retarding ceiling, fire prevention inspection, automatic de-watering, doorman, patrol, Video surveillance, alarm system, avoiding of water tubes in critical areas, highly sensitive early fire detection, up-to-date infrastructure and building plans, smoking ban
Demagnetization of magnetic data carriers -> data loss from magnetic data carriers	strong magnetic field	Magnetic shielding
Damage of data carriers strong light -> Data loss	strong light	Physical protection
heavy usage -> system breakdown or data loss	several regular users	Foresee heavy usage, load tests before release; redundancy, modularisation
backup restorage takes long or is complicated by design -> High effort for system recovery and long time of system non-availability	Developer, designer	Modularisation of system and backup
backup restorage takes long or is complicated because of data load -> High effort for system recovery and long time of system non-availability	maintainer	archiving of unused data

Undiscovered system breakdown -> System not available until breakdown is discovered and system recovered	administrator	monitoring

Sources: [Fir03c],[BSI04]

QA: Integrity in terms of Accuracy

Subfactors: accuracy = consistency between application and domain, i.e. timely accuracy (of time interval), OneToOneAccuracy (one object in the application corresponds to one and only one entity in the domain), ValueAccuracy, PropertyAccuracy, Consistency (external and internal) [CNYM00], p.165f

Threat -> consequence	**Misuser**	**countermeasure**
Use data format (type, dimension, initial value, default value, unit, resources, scope) inadequate for domain data -> stored data are useless or insufficiently accurate	Developer, requirements engineer	Systematic domain analysis, requirements engineering and system design
inaccurate calculations, calculation flaws -> Inaccurate results, in worst case useless	developer	programming rules (constraints on the programming process), developer training, testing
Other errors leading to inaccuracy -> Inaccurate data, in worst case useless	developer	attribute, from part to whole, explicit aggregation, accurate information reception, superclass, subset, conservation, subtype, attribute selection, derived info, correct information flow [CNYM00], p.168ff
Functional bugs in requirements and features: bugs having to do with requirements as specified or as	Requirements engineer	Good requirements engineering, including verification and validation

implemented, for instance the requirement or a part of it is incorrect, undesirable (requirement is correct as stated but it is not desirable), not needed, ambiguous; requirement is illogical (usually because of a self-contradiction), unreasonable (logical and consistent but unreasonable with respect to the environment and/or budgetary and time constraints), unachievable (requirement fundamentally impossible or cannot be achieved under existing constraints), Inconsistent, incompatible (with other requirements, with configuration or with environment) -> system does not correspond to customer needs		
Non-Verifiability of requirements: Unverifiable (the requirement, if implemented, cannot be verified by any means or within available	Requirements engineer	Good requirements engineering, including verification and validation

time and budget. For example, it is possible to design a test but the outcome of the test cannot be verified as correct or incorrect.), Untestable (it is not possible to design and/or execute tests which will verify the requirement. Untestable is stronger than unverifiable.), PRESENTATION (bugs in the presentation or documentation of requirements. The requirements are presumed to be correct, but the form in which they are presented is not. This can be important for test design automation systems which demand specific formats.), Presentation, documentation (format, media, etc.), presentation violates standards for requirements -> accuracy of system can not be verified		
Requirement changes: requirements, whether or not correct, have been changed (requirements changed or deleted, new	Requirements engineer	Requirements management, especially change management (detection and documentation of changes, impact analysis, communication of changes)

requirements added) between the time programming started and testing ended, Domain changes (input data domain modified: e.g., boundary changes, closure, treatment), changes to performance and other quality requirements (e.g., throughput) and/or timings. -> system does not correspond to customer needs		
Requirements implemented wrongly (coding, typography, standards violation), errors in component, interfaces or architecture -> system does not correspond to customer needs	developer	Quality management

Sources: [CNYM00], p.168ff; [BV00]
Metrics: relative deviation between system data and exact data; number of ciphers for input and output as well as for internal calculations

QA: Integrity in terms of Completeness

Threat -> consequence	**Misuser**	**countermeasure**
Sabotage or unintentional corruption of data by user error like input in wrong field, no input -> Data loss or	User, administrator	Usability of user interface, training and documentation, auditing, consistency checking, confirmation, cross examination, tracking assistance, time-efficiency of use cases, availability of

corruption		services, fault tolerance of user interfaces
Wrong system usage -> incorrectness of input (input into wrong fields, wrong input, missing input, incorrect saving of data), incorrectness of output (e.g. invoices sent to clients), user dissatisfaction, user refusal of the system, inefficiency of user tasks (if user actions must be repeated or data corrected), non-adherence to processes, incomplete accomplishment of a use case, no mental model of system	user, administrator, maintainer	Usability of the user interface
Wrong system usage -> damage like above	user, administrator, maintainer who suffer from too strict security policies and processes	Trade-off between security and usability requirements
Sabotage or intentional corruption of data or software, vandalism, fraud -> destruction; systematic trying of passwords, intentional effectuation of abnormal end, misuse of user rights, misuse of administrator rights, replay of	cyber-terrorist, corporate raider, vandal, other intruder	Definition and enforcement of security policies; foresee misuses in requirements analysis and system design; Identification, authentification, authorization, intrusion detection and intrusion response system, nonrepudiation, privacy/ confidentiality; Immunity, Integrity, survivability, physical protection, security auditing; emergency definition, emergency plan,

messages, Hoax, mail bombing, flooding -> Data or software loss or corruption		emergency responsible, emergency handbook, alarm plan, restart plan, emergency practice, substitution procurement plan, insurances, PC emergency/ recovery disk, escalation policy; runtime safety check, checks on the validity of input data, watchdog timers, delay timers, software filters, software-imposed initialization conditions, additional exception handling, assertion checking; authentication enforcement, auditing, consistency checking, cross examination, tracking assistance, certification, authorization, justification enforcement; security in general
System errors -> Data loss or corruption	developer	exception handling, resource assignment, validation, auditing, consistency checking, confirmation, cross examination, verification, check point, better information flow; recoverability
Incompleteness of requirements: the requirement as specified is either ambiguous, missing, incomplete, duplicated, overlapped, overly generalized (e.g., too powerful for the application), not downward compatible, insufficiently extendable or overly specified.	Requirements engineer	Good requirements engineering, including verification and validation

-> software does not correspond to customer needs, e.g. does not contain all necessary data or support not all necessary functionalities		
user error & bad reliability in terms of maturity and fault tolerance -> data loss	regular user	Foresee user errors by system design, fault detection, catching exceptions; formal methods for design; standard compliance
heavy usage -> system breakdown or data loss	several regular users	Foresee heavy usage, load tests before release; redundancy, modularisation
Bad recoverability of system after breakdown -> partial data loss	administrator	Recoverability, redundancy

Sources: [CNYM00], p.176ff; [BV00]

QA: privacy/ confidentially

Threat -> consequence	**Misuser**	**countermeasure**
Intentional unauthorized disclosure = access to or espionage of proprietary information, active wiretapping of wires, phone calls and data transfer, System penetration; manipulation of wires, systematic trying of passwords, IP spoofing, misuse of routing	An intruder who is not a regular user, e.g. professional criminals, hackers, crackers, industrial spies, foreign governmentals, foreign military,	Definition and enforcement of security policies; foresee misuses in requirements analysis and system design; Identification, authentification, authorization, intrusion detection and intrusion response system, encryption, security auditing, nonrepudiation, privacy/ confidentiality; Immunity, Integrity, survivability, physical protection, security auditing; emergency definition, emergency plan, emergency responsible, emergency handbook, alarm plan, restart plan, emergency

protocol, login bypass, network analysis tools, bugging of rooms, mascerading, analysis of the message flow, non-repudiation of a message, unauthorized copying of a data carrier, DNS spoofing, web spoofing, hijacking of network connections -> Hacker/ spy gets hold of data, maybe hands them on or publishs them		practice, substitution procurement plan, insurances, PC emergency/ recovery disk, escalation policy; runtime safety check, checks on the validity of input data, watchdog timers, delay timers, software filters, software-imposed initialization conditions, additional exception handling, assertion checking
Unauthorized Access by Insiders -> loss of privacy/ confidentiality, extortion, trespass	a user who executes a use case not intended for him/her, with different motivations: harmful goal, disgruntledness, or by error (just because it is possible)	foresee misuses in requirements analysis and system design; Identification, authentification, authorization, intrusion detection, nonrepudiation, privacy/ confidentiality, security auditing
Unintentional disclosure by user error -> (public) disclosure of data	User, administrator	foresee misuses in requirements analysis and system design; Identification, authentification, authorization, intrusion detection, nonrepudiation, privacy/ confidentiality, security auditing, training
Unintentional disclosure by software enhancement -> (public) disclosure of data	developer	programming rules (constraints on the programming process), developer training
software test with production data -	administrator	Software test with (realistic) test data; anonymization of

> Developers see confidential data		real data or systematic generation of test data
Social engineering	Hacker/ spy/ other users	User and administrator training; four eyes principle for access to confident data

Sources: [Fir03c], [BSI04], [Ric03], [Ran97]

QA: operational security

subfactors: identification, authentication, authorization, immunity, integrity, intrusion detection, nonrepudiation, privacy/ confidentiality, security auditing, survivability [Fir03c]; trust

Threat -> consequence	**Misuser**	**countermeasure**
probe (access a target in order to determine its characteristics), scan (access a set of targets sequentially in order to identify which targets have a specific characteristic)	Intruder, often preparing an attack	Foresee misuses in requirements analysis and design; security concepts and policies, e.g. authorization concept; requirements engineering of security requirements; security testing; user training
Neglect of security aspects during software engineering	Developer, customer	Foresee misuses in requirements analysis and design; security concepts and policies, e.g. authorization concept; requirements engineering of security requirements; security testing; user training
careless Internet usage and unintentional installation of malware, e.g. trojans, computer viruses -> Loss of data, integrity, privacy, availability, resources	users	Virus protection concept, choice and operation of anti-virus software, notification of virus infections, regular update of anti-virus software, regular virus scan
Uncontrolled but	Users	User training, authorization

intentional installation of software by users -> side-effects, security flaws		concept for installations
Hoax leads a user to detrimental manipulation of system	user	User training, user warning about actual hoaxes, authorization
breakdown of existing security devices by error -> Enabling of security threats which were impossible so far	administrator	Redundancy of security devices or detection of breakdown plus shutdown of system
wrong operating system version, incorrect system generation, or other host environment problem	administrator	Administrator training, testing

Sources: [Fir03c], [BSI89], [BSI04], [Ric03], [BV00], [HL98], [Lut93]
Metrics: mean time between break-ins, % break-ins foiled, cost of loss, insurance claims [SM98]; standard compliance (name of standard, level of compliance), type of auditing reports, auditing frequency, number of failed intrusion attempts, likelihood of breach, level of security, likelihood of accident, cost of accident [DPB+04] p.96; level of trust; encryption algorithm

Further details about computer security flaws can be found in [LBMC94].

QA: Immunity and Survivability

Threat	**Misuser**	**countermeasure**
All attacks from outside the system and inside -> damage: see "operational security"	intruder, user, administrator	All threats are foreseen and meet a countermeasures which prevents them
Neglect of immunity and survivability aspects during software engineering	Developer, customer	Foresee misuses in requirements analysis and design; requirements engineering of security requirements; security testing

Case study in section 6 was removed because it is confidential

Sources: [Fir03c]

QA: Suitability

Threat -> consequence	**Misuser**	**countermeasure**
Software engineering flaws -> System functionalities do not fit user needs and processes; Users do not use the system, use it wrong, missing or incomplete data	Developer, requirements engineer	Constraint on software development process: good/ formal requirements engineering and system design; (acceptance) testing
User does not understand the system -> user can not profit of all advantages of the system	user, administrator, maintainer	Provide documentation and training, feedback, metaphors, use rules and standards like ISO 9241 for interface design
Processes change -> System functionalities do not fit user needs and processes any longer	maintainer	Change management; regular review
changes to the system + bad maintainability of system (badly analysable/ changeable) code, bad stability or testability -> High maintenance cost; system does not work as before change	maintainer	maintainability compliance, constraints on the software development process (e.g. coding guidelines for assuring analysability, changeability, stability and testability), e.g. encapsulation/ modularity, structuredness, reduction of complexity, tracing, reuse, separation, indirection, abstraction, location transparency, layered architecture, moduls model + view + controller, publish-subscribe, traceability matrix, interface consistency
Changes to the system + system	maintainer	conformance to standard, encapsulation of

not portable or maintainable -> loss of functionality of the system, i.e. new configuration does not work any more		functionality, developer training, independence, modularity, operating system functionality

QA: Interoperability

Threat -> consequence	**Misuser**	**countermeasure**
Misunderstandings about semantics of interface data	Requirements engineer	Describe context of application; interface description must contain information about semantics; workshops, requirements and design reviews
Interface to other system is not or hardly possible to be developed for technical reasons (e.g. no API included) -> Interface does not work	designer	Choose another technology or accept constraints on system use
bugs in the interface to third-party software or other software developed externally (due to a misunder-standing or wrong interpretation of the features and operation of the third-party software; or due to flaws in the third-party software which the vendor does not correct) -> interface does not work, threat to	Third-party developer	Interface specification and testing

time efficiency and data correctness/ integrity		
System not compliant to legal or domain standards or conventions about functionality, development process and documentation	Requirements engineer, developer	Do research and analysis, Developer training, coding standards

Sources: [BV00]; standard interfaces

QA: Reliability in terms of Maturity and Fault tolerance

Threat -> consequence	**Misuser**	**countermeasure**
involuntary creation of undefined system state during normal use -> Inacceptable system slowdown or breakdown or data loss	regular user	Foresee all possible system states in specification, catching exceptions; formal methods for requirements engineering and design
bad interoperability of interface to other system -> inreliable interface, i.e. threat to time efficiency and data correctness	developer	developer training, adherence to interface standards, tests before release
Changes to the system + system not portable or maintainable -> loss of reliability of the system	maintainer	conformance to standard, encapsulation of functionality, developer training, independence, modularity, operating system functionality

Sources: [DPB+04], p.40
Metrics: mean time between failures, failure per unit time, degrees of failure (severe, tolerable, etc.) [SM98], relative uptime, maximum downtime per failure, mean time between failures, time in operation, number of users, estimated remaining faults, % of faults leading to

failure, number of interface infringements per use case or system task, likelihood of information loss, critical information, interface state recover, execution state recover, actions that need to be performed, failure recovery time [DPB+04], p.40

QA: Reliability in terms of Recoverability

Application to exploratory scenario: system breakdown and recoverage by administrator

Threat -> consequence	**Misuser**	**countermeasure**
No backup -> system is not recoverable after breakdown	Administrator who does not know how to do a backup	training, automated backup facility, control
No backup -> system is not recoverable after breakdown	Administrator with unreliable work style	Four-eyes-principle, control, automated backup facility
No backup and backup restorage foreseen in system -> system is not recoverable	Developer or requirements engineer	Define use cases “backup” and “backup restoring“ during requirements engineering
backup restorage does not work -> backup can not be recovered when needed	administrator	Regular tests of recovery scenario, sanity check, redundant backups and restorage facilities
backup restorage takes long or is complicated by design -> High effort for system recovery and long time of system non-availability	Developer, designer	Modularisation of system and backup
backup restorage takes long or is complicated because of data load -> High effort for system recovery and long time of system non-availability	maintainer	archiving of unused data

Sources: [DPB+04], p.50ff
Metrics: fault detection time, fault recovery time, repair time, inspection efficiency, complexity, coupling, fault density, testing effort [DPB+04], p.50ff

QA: Usability (Understandability, Learnability, Operability)

Subfactors of usability: effectiveness, satisfaction, experience, productivity, safety, error tolerance, learnability, usability compliance, understandability, attractiveness, operability [DPB+04] p.75

Threat -> consequence	**Misuser**	**countermeasure**
Software engineering does not take usability important -> users don´t use system, work inefficiently, lack of data integrity	developer	constraints on the interface definition: understandability, learnability and operability of user interface; usability test
Software engineering does not consider common standards -> system does not comply to user´s expectations -> users don´t use system, work inefficiently, lack of data integrity	developer	constraints on the interface definition: use rules and standards like ISO 9241 for interface design; conformance and conformance review

Sources: [DPB+04] p.75
Metrics: number of user errors, task completion times, task performance, training time [SM98]; % of goals achieved, % of users successfully completing a task, suggestions for improving the product, comments on preference of version A versus version B, rating scale for satisfaction, frequency of disrectionary use, number of negative references, rating of product, frequency of complaints, user effort/ time, cost of usage, used material, productive period [DPB+04] p.75

QA: Time Efficiency

Subfactors: response time, throughput [CNYM00], p.220

Threat -> consequence	**Misuser**	**countermeasure**
usage of a component by too many users or requests -> slowdown of	user or user groups, maintainers and administrators who are obliged to do the same task at the	observation of arrival patterns; constraints or loosening of constraints on system use

communication or processing	same time, e.g. produce the same individual report on Friday afternoon at 4 pm or to log in by work or shift start	
inappropriate hardware, network or software -> slowdown of communication or processing	user or user groups, maintainer and administrator	performing load tests before release; observation of execution time; identification and improvement of bottleneck components; constraints on hardware or software; locality, parallelism, spare schedule; caching, sharing
denial-of-service attack, mail bombing, flooding -> System too busy with attackers request to operate normal requests	Intruder like hacker, cracker, cyber-terrorist, criminal	Security measures like authorization, firewall, etc.
Performance parasites: any bug whose primary or only symptom is a performance degradation: e.g., the harmless but needless repetition of operations, fetching and returning more dynamic resources than needed -> decrease of performance	developer	performing load tests before release; observation of execution time
Interface to other system is inefficient (bad interoperability) -> Threat to time efficiency and data correctness	developer	developer training, adherence to interface standards, tests before release
user error & bad reliability in terms of maturity and fault tolerance -> Inacceptable system slowdown	regular user	Foresee user errors by system design, fault detection, catching exceptions; formal methods for design; standard compliance

or breakdown		
heavy usage -> Inacceptable system slowdown or breakdown	several regular users	Foresee heavy usage, and load tests before release; redundancy, modularisation
Changes to the system + system not portable or maintainable -> loss of efficiency of the system, i.e. communication is slowed down	maintainer	conformance to standard, encapsulation of functionality, developer training, independence, modularity, operating system functionality

Sources: [DPB+04] p.62, p.68; [BV00]
Metrics: transaction volumes, response times [SM98], boot/ start time, shutdown time [DPB+04] p.62; duration of a use case

QA: Resource Efficiency

Subfactors: main memory, secondary storage [CNYM00], p.220

Threat -> consequence	**Misuser**	**countermeasure**
Heavy load -> memory overflow, error messages to users, inefficient work, data loss, lowered bandwidth	user or user groups, maintainers and administrators who are obliged to do the same task at the same time, e.g. produce the same report on Friday afternoon at 4 pm or to log in by work or shift start	observation of arrival patterns; constraints or loosening of constraints on system use
denial-of-service attack -> Breakdown of system	intruder	Security measures like authorization, firewall, etc.
inadequate resources like hardware (processor, memory, secondary storage), network or software -> memory overflow, error	Developer, maintainer and administrator	constraints on workload distribution, capacity of memory/ network/ processor/ etc, type and position of devices, sharing, locality, early fixing (static offset determination, uncompressed format, indexing), late fixing (reduce run time

messages to users, inefficient work, data loss, lowered bandwidth		organization, dynamic offset determination, compressed format), execution ordering/ prioritization methods, layering
Partitions and overlays: memory or virtual memory is incorrectly partitioned, overlay to wrong area, overlay or partition conflicts -> decreased resource efficiency	Administrator who did the partitioning	training
Performance parasites: any bug whose primary or only symptom is a performance degradation: e.g., Memory leak, fetching and returning more dynamic resources than needed -> decrease of performance	developer	performing load tests before release; observation of resource need
Changes to the system + system not portable or maintainable -> loss of resource efficiency of the system	maintainer	conformance to standard, encapsulation of functionality, developer training, independence, modularity, operating system functionality

Sources: [DKVP03], [DPB+04] p.62, [CNYM00], p.225 and p.237; [BV00]
Metrics: workload distribution, resource consumption (% of usage), cost of memory, number of network nodes [DPB+04] p.62

QA: Maintainability (Analysability, Changeability, Stability, Testability)

Further subfactor: installability
Application to growth scenario: changes to the system by maintainer

Threat -> consequence	**Misuser**	**countermeasure**
maintainability was no focus during systems engineering -> maintainability is low and/ or degrades by time	Developer	Specify maintainability requirements; test maintenance scenarios; keep to coding standards
changes to the system + maintainer error (e.g. change without former impact analysis; making an error during the impact analysis) -> High maintenance cost; system does not work as before change	maintainer	Training and documentation, suitable constraints on the maintenance process (instructions demanding that and how impact analysis has to be done), tool support
changes to the system + insufficient re-test of functionalities -> system does not work as before change; threat to functionality, reliability and efficiency of the system	maintainer	automatic test case generation, testsuite interface generation
Erroneous installation -> Installation must be re-done, or later problems in operation	maintainer	Constraints on installation procedure
bugs in the documentation associated with the code or the content of comments contained in the code: Incorrect/ wrong, inconsistent (with itself or with	Developer	Documentation standards and documentation review

other statements), incomprehensible (documentation cannot be understood by a qualified reader), incomplete (important facts are missing), missing: major parts of documentation are missing -> more effort for maintenance, errors		
Interface to other system is badly maintainable because of poor code or missing/ poor documentation (bad interoperability) -> High effort for maintainer, error proneness	developer	Interface documentation, adherence to standards

Sources: [DPB+04], p.17 and 25; [BV00]
Metrics: time needed to perform task, actions to be performed, number of supported users (scalability), number of unexpected behaviours (stability), affected components, cohesion, coupling, size, comment frequency, complexity, depth of inheritance tree, number of children [DPB+04], p.17 and 25ff

QA: Portability (Adaptability, Installability, Replaceability)

Subfactors: adaptability, installability, replaceability, co-existence, standard compliance, environment compatibility [DPB+04], p.86
Application to growth scenarios: changes in the environment of the system, e.g. adapt the system to changes in interfaces, install on other platform (hardware, operating system, etc.; transfer and configure), replace part of the system by another; all done by administrator

Threat -> consequence	**Misuser**	**countermeasure**
Portability was no focus during systems engineering ->	Developer	Specify portability requirements; test portability scenarios; keep to coding standards

portability is low and/ or degrades by time		

Sources: [DPB+04], [DPB+04], p.86
Metrics: number of platforms the system can be ported to, effort and cost required for each system porting [SM98]; time needed to perform user task, steps to be performed for user task, removable components, backward compatibility, external communication complexity of system to be replaced, compliance to standard [DPB+04], p.86

QA: Compliance

Threat -> consequence	**Misuser**	**countermeasure**
Compliance was not designed into the system during its development	Developer	Specify compliance requirements; review compliance; legal or domain standards or concentions about functionality, development process and documentation

Sources: [DPB+04]
Metrics: name of standard and level of compliance

A.5 Assets with lists of vulnerabilities and countermeasures

Each asset has a number of properties which might be misused and therefore called a vulnerability. The most extensive list of assets and their vulnerabilities we found at [BSI04]. These vulnerabilities were given for assets on different levels of granularity: some for data bases in general, some for specific data base programs, etc.. Of course, a specific data base program shows the same vulnerabilities as all data bases do in general. Therefore it seemed logic to organize the assets in a hierarchy as shown in figure 7. This hierarchy certainly is not a complete presentation of all possible assets, but structures the content of [BSI04] and of this section. Read it like this: under the title “computer networks” we list the vulnerabilities applying to computer networks in general. Under “mobile devices” we will only name those specific to mobile devices, but additionally those for computer networks are relevant also. So you find the vulnerabilities for an asset by taking together all those of the whole branch. It is the same for the countermeasures.
On the top level, we start with the categories of Firesmith: data, communications, services, hardware components, personnel [Fir03c].

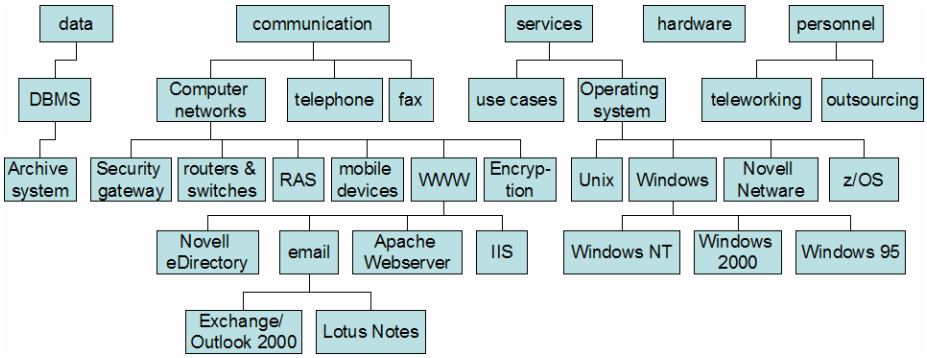

figure 4: hierarchy of assets as used in this section

For the sake of shortness, we will not name each vulnerability – countermeasure – pair explicitly. Where for a given vulnerability the corresponding countermeasure is obvious, we do not name it here. Many countermeasures counteract against several vulnerabilities at a time and these (and only these) are summed at the end of the list.

Data:

- Are manipulated by users who might fail
- Are manipulated automatically by interfaces to other systems
- are only a model for domain data
- must have a defined format in the system
- data are business values
- change steadily
- information monitoring
- information aggregation
- information storage -> countermeasure: data description
- information transfer
- information collection
- information personalization
- copyright
- unsecured paper file and data carrier transport
- transport of data carrier by postage -> countermeasure: sufficient labelling and safe packaging of data carriers
- use of mobile data carriers -> countermeasure: appropriate storage of data carrier
- insufficient storage media for emergency
- export or import of data
- transfer of wrong or unwanted data
- unstructured data storage
- data backup
- defect data carriers
- remote access/ remote maintenance via modem

- **countermeasures** for data vulnerabilities:
  - appropriate and controlled disposal of protectable resources
  - commitment of employees to backup rules for data and data carrier exchange
  - secure deletion of data and disposal of data carriers
  - rules for removal of data carriers
  - training of personnel with respect to controlled exchange of data carriers
  - physical deletion of data carriers before and after use
  - virus scan at each data carrier exchange and data transfer

- encryption, checksums and digital signatures for data transfer
- verification of data before transfer
- choice of appropriate postage way for data carriers
- data backup and recovery plan
- data backup on PC
- appropriate storage of backup data carrier
- documentation of backup
- regular backup of server hard disk
- choice of appropriate data formats
- comply to legal regulations for protection of personal data and telephone data

Communications (computer networks):

- configuration of a network
- interfaces to other systems
- might pass over public network
- limited bandwidth
- peer-to-peer connections
- network might break down
- insufficient dimensioning of line
- evaluation of protocol data
- integration of DOS PCs in a server based network
- limitation of the speed of transfer or processing
- uncontrolled communication connections
- inadequate configuration of active network components
- incompatible active and passive network components
- conceptional weaknesses of the network
- exceeding of allowed wire or bus length of ring size
- missing or insufficient strategies for network and system management
- breakdown or interference of network components
- breakdown of system management system
- complexity of configuration
- insecure protocols
- unsecured connections
- treatment of ICMP at security gateway
- erroneous time synchronization
- insufficient identification check of communication partners
- complexity of access possibilities to networked IT systems
- not disconnected connection
- bad or missing authentication
- unprotected wires
- temporarily freely accessible accounts
- ex post modification of information
- elements for network coupling
- internal remote access
- external remote access

- **countermeasures** for communication/ network vulnerabilities:

    - planning and conception

- security policy
- administrator training
- monitoring of existing connections
- documentation of configurations
- control of protocol files
- firewall
- package filter
- choice and operation of filter rules
- security policy for use of peer-to-peer services and control
- analysis of actual state of network
- network concept
- network management concept
- appropriate choice of network protocol
- network management tool
- access restriction to accounts and terminals
- locking or deleting of not required accounts and terminals
- virus scanning of incoming files
- logging of firewall activities
- audit and monitoring of network activities
- restrictive communication via package filter to minimum
- firewall
- encryption
- LDAP Services for NDS
- appropriate choice of network topography
- appropriate choice of wire
- documentation and labeling of wires
- obligatory network password
- locking of server console
- appointment of an additional network administrator
- overview over network services
- local NTP server for time synchronization
- compatibility check of sender and receiver system
- use of one-time passwords
- use of security mechanisms of UUCP
- limitation of peer-to-peer functionalities in server-based network
- choice of appropriate backbone technology
- appropriate physical segmentation
- appropriate logical segmentation
- use of time stamp service
- encrypted communication
- NAT (Network Address Translation)
- deactivation of not required network services
- safe connection of external network via Linux FreeS/WAN
- integration of DNS server with security gateway
- overview over availability requirements
- copy of transmitted data
- emergency plan
- redundancy of network components and communication connections
- regular backup of configuration data of active network components
- authentication

Services:

- authorization
- accessibility of web-services, no strict protection by firewall possible
- processes are defined by domain
- IT security management
- definition of access rights
- administration of access rights
- disclosure of software vulnerabilities
- undocumented functions
- coding faults introduced during software development
- improper software installation
- when a program is used in an environment for which it was not intended
- memory leak
- exception handling
- computers also run malware like viruses or trojans

- **countermeasures** for service vulnerabilities:

    - use of security mechanisms of the application software
    - upgrade and update of software and hardware
    - one service per server
    - redundancy of software
    - copy of used software
    - prompt installation of patches and updates

Hardware components:

- physical limits, e.g. can break down when getting too hot
- resources (e.g. memory capacity) are limited
- could change place of usage
- dependence on power supply -> countermeasure: emergency power supply supply, segmentation of power circuits,
- dependence on power distributor
- discharged or out-dated emergency current supply
- data carrier not available at due time
- insufficient labelling of data carriers
- delivery of data carriers
- use of undeclared components
- sharing of directories or printers
- missing or inadequate segmentation
- outsourcing of maintenance
- maintenance down-times

- **countermeasures** for hardware component vulnerabilities:
    - choice of hardware and wires
    - appropriate choice of place of usage
    - monitoring of components
    - physical protection
    - emergency off switch

- redundancy
- documentation
- secure relocation
- system management policy
- system management tool
- overview over existing IT systems
- infrastructure maintenance
- permission process for new IT components
- test of new hardware and software
- change management
- secure access for local administration and remote access
- documentation of capacity needs of applications

Personnel:

- no or poor training
- no or poor documentation
- low intelligence
- helpful (facilitates social engineering)
- impatient
- click even on functionalities and buttons they are not to use
- curious
- overworked
- work under time pressure
- unfavourable work conditions
- can be unsatisfied with their employer
- limited personnel resources
- missing or insufficient rules
- insufficient knowledge about rules
- ignorance of IT security rules
- insufficient control of security measures
- inadequate reaction to security incidents
- negligent/ convenient, e.g. storage of passwords
- missing/ improper/ incompatible equipment
- no or insufficient servicing
- insufficient adaptation on changes in IT use
- alternation of users
- inadequate limitation of user environment
- complexity of software configuration
- install software
- cleaning, service and other external personal -> countermeasures: clear organizational rules for maintenance and repair works
- Handling of passwords
- Easy access to building
- Unclear responsibilities

- **Countermeasures** against personnel vulnerabilities:
  - Training
  - Documentation
  - Auditing
  - guidance by system (e.g. automatization)

- [AER02]: notice/ awareness, choice/ consent, assess/ participation
- clear rules and responsibilities
- separation of roles
- support and helpdesk for IT users
- appointment of an administrator and a deputy
- documentation of changes on system
- information about security and other vulnerabilities of system
- split-up of administrator tasks
- the tidy desk
- reaction to violations of security policy
- standardization of work places
- documentation of approved users and their rights
- definition of roles
- set up of a limited user environment
- process for software acceptance and release
- creation of a requirements document for and choice of standard software
- testing of standard software
- installation instructions and configuration for standard software
- procurement of certified software
- license administration and version control for standard software
- control of delivery
- management reports about IT quality
- documentation of quality securing process
- policy for access control
- change management
- attribution of responsibility for information
- applications and IT components
- controlled initial training/ briefing of new employees
- commitment of employees to observing relevant laws
- instructions and regulations
- rules for deputyship
- controlled process for leaving employees
- contact person for personal problems
- avoiding of disturbance of working atmosphere
- ergonomic work place
- commitment of users to logoff from system after task completion
- software reinstallation on workplace computers
- security check for employees
- screen-lock
- post-processing of incidents
- respect of legal frameworks

Operating system:

- Planning of use
- Installation
- Configuration
- De-installation of operating system
- Migration

- **Countermeasures** for operating system vulnerabilities:

- user training
- administrator training
- security rules
- security audits
- definition of standards for configuration
- monitoring
- maintenance
- system management
- backup
- recovery mechanisms
- use of security mechanisms of NFS
- use of security mechanisms of NIS
- use of BIOS security mechanisms
- minimum operating system
- deactivation of DNS
- secure BIOS update

Windows NT:

- administration rights on Windows NT system
- sharing of folders
- Migration from Windows NT to Windows 2000
- Integration of DOS computers into Windows NT network
- Configuration of TCP/IP network administration
- Configuration of remote access

- **countermeasures** for Windows NT vulnerabilities:
  - planning of Windows NT network
  - definition of security policy for Windows NT Client-Server network
  - control of network
  - password protection
  - structured system management
  - user profiles to limit use options
  - device protection logging
  - securing of boot process
  - restrictive allowance of access rights on files and folders
  - deactivation of automatic CD ROM recognition
  - protection of administrator accounts
  - generation of rescue disk

Windows 2000:

- planning of Windows 2000 use
- Configuration of Windows 2000 as domain controller
- Configuration of Windows 2000 as server
- Configuration of Windows 2000 services
- Configuration of Windows 2000 as workstation
- Configuration of secure channel
- Configuration of DDNS
- Configuration of WINS
- Configuration of DHCP

- Usage of CA and Windows 2000 CA structure
- Usage of EFS
- Usage of IPSec
- sharing of files and folders

- **countermeasures** for Windows 2000 vulnerabilities:
  - planning of group guidelines
  - password protection
  - securing of boot process
  - device protection, protection of registration
  - deactivation of automatic CD ROM recognition
  - generation of rescue disk

Windows 95:

- convertion of file names when storing under Windows 95
- storage of passwords in Windows 95
- sharing of folders
- networked Windows 95 computers

- **countermeasures** for Windows 95 vulnerabilities:
  - setup of user profiles
  - system policy for limiting of use
  - deactivation of automatic CD ROM recognition
  - use of login password
  - generation of rescue disk

z/OS:

- character conversion when using z/OS -> countermeasure: user and administrator information
- login process
- configuration of z/OS operating system
- configuration of z/OS web server
- file protection in z/OS system 81
- system time on z/OS system
- configuration of z/OS security system RACF
- operation of z/OS system functions
- protection of z/OS system configuration against dynamic changes
- control of batch jobs in z/OS -> batch job planning
- RACF attributes
- start process
- security critical z/OS service programs
- unix system services on z/OS systems
- z/OS trace functionalities
- transaction monitors

- **Countermeasures** for z/OS vulnerabilities:
  - use of restrictive user names
  - use of the security system RACF
  - use of RACF exits
  - synchronization of passwords and RACF commands

- use of machine-oriented z/OS terminals
- securing of Linux for zSeries
- definition of system limits of z/OS
- workload management
- license key management
- Parallel-Sysplex on z/OS
- use of VTAM Session Management Exit
- training on Linux and z/VM for zSeries systems

Unix:

- missing authentication possibility between X servers and X clients
- administrator rights
- uucp
- integration of DOS PC into Unix network
- protocols and services

- **countermeasures** for Unix vulnerabilities:
  - split-up of administrative tasks
  - obligatory password protection
  - restrictive attribute allocation for Unix system files and folders
  - restrictive attribute allocation for Unix user files and folders
  - secure execution of executable files
  - logging
  - use of security mechanisms of sendmail, rlogin, rsh and rcp
  - secure operation of telnet, ftp, tftp und rexec
  - use of secure shell
  - one-side connection
  - setup of Closed User Group

Novell Netware:

- missing or inadequate activation of Novell Netware security mechanisms
- complexity of NDS -> countermeasure: design of concept
- migration of Novell Netware 3.x to Novell Netware Version 4.x
- "Hacking Novell Netware”
- administrator rights under Novell Netware 3.x

- **countermeasures** for Novell Netware vulnerabilities:
  - secure installation of servers
  - appropriate configuration of servers
  - secure operation of servers and networks
  - revision of servers and networks
  - renouncement on activation of remote control
  - design of time synchronisation concept
  - documentation of Novell Netware Networks
  - C2 security under Novell 4.11
  - DHCP server under Novell Netware 4.x
  - simplified and more secure network management with DNS server under Novell Netware 4.11

Security Gateway:

- installation and configuration of security gateway
- incident precaution concept for security gateway

- **countermeasures** for security gateway vulnerabilities:
  - concept for/ choice of/ appropriate configuration and operation of security gateway
  - integration of servers in security gateway
  - secured placement out of operation or replacement of components of a security gateway
  - outsourcing of security gateway
  - high availability of security gateway
  - integration of proxy server with security gateway
  - integration of Virtual Private Networks with security gateway
  - logging of security gateway
  - integration of virus scanner with security gateway
  - emergency plan for security gateway

Routers and Switches:

- default configuration on routers and switches
- configuration of routers and switches: local basis configuration and network configuration
- administration of routers and switches
- remote access for management tasks at routers
- routing protocols

- **countermeasures** for router and switch vulnerabilities:
  - appropriate choice of routers and switches
  - secure placing out of operation
  - configuration checklist
  - logging
  - protection of switch ports
  - setup of access control lists on routers
  - data backup and recovery

WWW use:

- out-dated or wrong information on a website
- error at the request for and management of internet domain name
- administration of internet domain
- installation of internet PCs
- eCommerce
- execution of active content possible

- **countermeasures** for WWW vulnerabilities:
  - concept for WWW use
  - secure operation of www server
  - appropriate internet service provider

- www editorial team
- protection of www files
- security of web browers
- use of stand-alone systems for internet usage
- choice of appropriate modem
- appropriate physical position of modem
- safe administration of modem
- regulation of modem use
- training of users to modem usage
- appropriate modem configuration
- personal firewall for internet PCs
- secure connection of internet PCs
- integration of web server with security gateway
- integration of a web application with web application server and database server with security gateway
- backup for internet PCs
- emergency plan for web server
- use of S-HTTP
- use of SSL

Novell eDirectory:

- installation of Novell eDirectory
- installation of Novell eDirectory client software
- deficient or insufficient planning of partitioning and replication of Novell eDirectory
- deficient or insufficient planning of LDAP access on Novell eDirectory
- configuration of Novell eDirectory
- configuration of Novell eDirectory client software
- assignment of access rights in Novell eDirectory
- configuration of the intranet client access to Novell eDirectory
- configuration of the LDAP access to Novell eDirectory
- breakdown of Novell eDirectory
- use in intranet
- use in extranet

- **countermeasures** for Novell eDirectors vulnerabilities:
  - training for use of client software
  - setup of access rights
  - use of LDAP access on Novell eDirectory
  - monitoring
  - secure communication
  - emergency plan for breakdown of Novell eDirectory service
  - data backup

Email:

- insufficient time authenticity of e-mail
- disordered email usage
- configuration of email server

- configuration of email client
- feigning of wrong sender
- alias files
- mailing lists
- limited capacity for incoming e-mails
- active content
- address books
- use of web mail
- dial-in numbers/ dialer liable to costs

- **countermeasures** for email vulnerabilities:
  - policy for email usage
  - virus and spam filter
  - regular deleting of emails
  - standardized email addresses
  - choice of email provider
  - rules for deputyship of email users
  - role related email addresses
  - secure operation of mail server
  - email encryption
  - email scanner on mail server
  - securing of emails by SPHINX (S/MIME)
  - integration of email server in security gateway
  - backup and archiving of emails
  - use of GnuPG or PGP

Exchange/Outlook 2000:

- migration of Exchange 5.5 to Exchange 2000
- access rights on Exchange 2000 objects
- browser access at Exchange
- connection of other e-mail systems to Exchange/Outlook
- configuration of Exchange 2000 Server
- configuration of Outlook 2000
- installation

- **countermeasures** for Exchange vulnerabilities:
  - administrator training about system architecture and security
  - user training about security mechanisms
  - secure operation of Exchange/Outlook 2000
  - monitoring and logging
  - emergency plan for breakdown of server
  - use of SSL/TLS
  - use of encryption and signatures for communication

Lotus Notes:

- configuration of Louts Notes Server
- configuration of the browser access on Lotus Notes
- active content at access to Lotus Notes
- "Hacking Lotus Notes"

- **countermeasures** for Lotus Notes vulnerabilities:
  - planning of operation
  - planning of domains and certificate hierarchy
  - planning and configuration of operation in intranet or DMZ with and without browser access
  - training for administrators about system architecture
  - user training about security mechanisms
  - secure installation
  - secure configuration of server
  - access restrictions for server
  - configuration of access lists on Lotus Notes databases
  - configuration of access rights on name and address book
  - activation of SSL protected browser access
  - configuration of authentication mechanisms for browser access
  - configuration of clients
  - browser configuration
  - handling of notes ID files
  - monitoring of system
  - encryption of Lotus Notes databases
  - creation of new Lotus Notes databases
  - encrypted communication
  - encrypted email
  - encrypted browser access

IIS:

- integration of IIS in system environment
- configuration of operating system for IIS
- configuration of IIS
- insufficient knowledge about vulnerabilities of IIS and test tools
- administrator and user accounts
- known vulnerabilities

- **countermeasures** for IIS vulnerabilities:
  - preparation of installation and secure configuration of Windows NT/ 2000
  - secure configuration
  - choice and configuration of authentication method for web offers
  - protection of critical files
  - operation of IIS in separate process
  - monitoring, deactivation of not required services
  - securing of virtual folders and web applications
  - deletion of example files and administration scripts
  - deletion of FrontPage Server enhancement
  - protection against unauthorized programme calls
  - deletion of RDS support
  - deletion of not required ODBC drivers
  - installation of URL filter
  - deletion of network shares
  - configuration of TCP/IP filter
  - prevention of SYN attacks

- deletion of not trustworthy root certificates
- emergency plan´
- backup

Apache Webserver:

- installation and configuration
- incident precaution concept
- configuration of operating system on Apache web server
- specific vulnerabilities

- **countermeasures** for Apache vulnerabilities:
  - planning of SSL operation
  - configuration of operating system
  - secure installation
  - secure configuration
  - configuration of access control
  - secure operation
  - server enhancement for dynamic web sites
  - installation of web server in chroot cage
  - use of SSL
  - emergency plan and power supply

Mobile devices (computers, telephones, PDAs):

- synchronisation of mobile devices
- user change of mobile computers -> countermeasures: controlled handing over and withdrawal of mobile devices, software reinstallation at change of user
- dependence on availability of mobile radio network
- insufficient security mechanisms of PDAs
- mobile phones and other mobile devices can be used for bugging
- limited battery capacity
- Calling Line Identification by use of mobile phone
- connection data of usage of mobile phone
- motion profile by use of mobile phone
- photo and video taking with mobile devices
- crash due to maintenance error
- efficiency -> spare resources, light-weight installation
- concurrent applications -> prevent additional installations by user

- **countermeasures** for mobile device vulnerabilities:
  - security policy for use
  - adequate keeping of mobile devices in mobile and stationary use
  - accumulative keeping of several mobile computers
  - theft protection devices
  - notice of loss
  - locking of mobile phone when lost
  - setup of mobile phone pool
  - securing of energy supply during mobile operation
  - use of security mechanisms of mobile devices (e.g. PIN, encrypted communication)
  - secure data transfer

- emergency plan
- backup
- central administration of PDAs
- password protection
- encryption
- alternative device like terminals

RAS System:

- erroneous administration
- inadequate usage of authentication services at remote access
- maloperation at use of RAS services
- configuration of RAS Client
- inadequate equipment of the working environment of RAS clients
- deactivation of security mechanisms for RAS access
- usage of RAS client as RAS server
- permission of external use of RAS components

- **countermeasures** for RAS vulnerabilities:
  - appropriate choice
  - concept for RAS use
  - appropriate installation and configuration
  - secure operation
  - use of authentication server
  - secure configuration under Windows 2000
  - emergency plan for RAS system

Encrytion:

- key management
- configuration of crypto modules
- operation of crypto modules
- availability of crypto module
- security of cryptographic algorithm
- correctness of encrypted data
- disclosure of cryptographic key
- counterfeited certificates
- interfaces of crypto modules
- physical security of crypto modules
- operating system
- irradiation security
- backup
- Application of crypto modules on different layers of the OSI reference model

- **countermeasures** for encryption vulnerabilities:
  - concept for encryption
  - appropriate choice and update of cryptographic method and product
  - rules for deployment of encryption

Telephone:

- Crosstalk
- maloperation of answerphone
- data saved in telephone system
- limited capacity of phone answering machine
- simple phone PIN code
- remote enquiry
- remote access for management tasks in telephone system and ISDN coupling elements -> countermeasures: renouncement or administration of authorization
- manipulation via ISDN-D channel possible
- telephone system interfaces

- **countermeasures** for telephone vulnerabilities:
  - appropriate physical position of telephone system and answerphone
  - documentation and helpdesk for users and telephone administrator
  - acquisition/ choice of appropriate telephone server & telephone & ISDN cards & answerphone
  - use of secure phone PIN code
  - evitation of critical information on answerphone
  - regular playback and deleting of stored messages
  - limitation of time of speech
  - documentation of ISDN card configuration
  - aspects of data protection for logging
  - deactivation of not required functionalities of ISDN card or ISDN router
  - use of existent security mechanisms of ISDN components
  - use of D channel filter, information of users about warning messages
  - symbols and tones
  - information of users about dangers
  - user training for telephone and answerphone use
  - logging of administrator works on telephone system
  - revision of telephone system configuration (comparison of as-is and to-be)
  - change of default passwords
  - protection of telephone server
  - password protection for telephones
  - deactivation of not required functionalities
  - regular backup of telephone system configuration data
  - basis phone number for emergencies and catastrophes
  - switching off of answerphone when being present
  - authentication via CLIP/COLP (CLIP= Calling Line Identification Presentation, COLP= Connected Line Identification Presentation)
  - Callback based on CLIP/COLP
  - emergency plan for breakdown of telephone system

Fax:

- bleaching of some fax papers -> countermeasure: photocopying of incoming faxes
- legal bindingness of fax
- disordered fax usage
- unauthorized use of a fax device

- unauthorized reading of incoming faxes
- analyze of rest information on fax devices
- feigning of a wrong fax sender
- intentional re-programming of target keys of a fax device
- limited capacity for incoming fax messages
- fax server
- overload of fax server
- administration of fax address books and mailing lists

- **countermeasures** for fax vulnerabilities:
  - security policy for fax usage
  - appropriate physical position of fax device
  - appointment of a fax responsible person
  - appointment of authorized fax operators
  - appropriate fax devices and fax server
  - appropriate disposal of consumable fax material and spare parts
  - provision and control of consumable fax material
  - turning off fax device off office hours
  - setup of a fax post office
  - regular controls of security policy
  - user information
  - blocking of certain fax numbers
  - locking of certain sender fax numbers
  - deactivation of not required functionalities
  - fax device with automatic enveloping of incoming faxes
  - use of appropriate fax title page
  - use of send and receipt protocols
  - telephonic announcement of a fax message
  - telephonic verification of correct fax receipt
  - telephonic verification of correct fax sender
  - control of saved fax addresses and protocols
  - activation of callback option
  - trader address list for fax re-acquisition
  - emergency plan for fax server breakdown

DBMS:

- complexity of DBMS
- missing or inadequate activation of database security mechanisms
- complexity of database access
- inadequate organization of alternation of database users
- administration
- breakdown of data base
- undermining of access control via ODBC
- limits of storage medium
- loss of data base integrity or consistency

- **countermeasures** for DBMS vulnerabilities:
  - appropriate choice of DBMS software
  - installation and configuration
  - security concept

- prevention of inference
- appropriate choice of physical position
- access control
- split-up of administration tasks
- rules for setup of users and user groups
- control of protocol files
- rules for database enquiries
- secured data export/ import
- structured data management
- blocking and deleting of not required database accounts
- securing of consistent database administration
- monitoring
- restrictive handling of database links
- encryption of data base
- integration of database server with security gateway
- rules of conduct after loss of database integrity
- backup
- archiving
- recovery mechanisms for database
- installation of ODBC drivers

Archive Systems:

- migration
- option of revision
- order criteria
- capacity of data carriers
- documentation of archive access
- transfer of paper data into electronic archives
- refreshing of data content
- refreshing of digital signatures
- execution of revisions
- destruction of data carriers at archiving
- planning of the physical place of archive systems
- use of adequate data carriers for archiving
- legal frameworks when using archive systems
- delayed archive information
- synchronization of index data when archiving

- **countermeasures** for archive system vulnerabilities:
  - market study for archive systems
  - appropriate choice of archive system
  - data formats and media
  - appropriate storage of archive media
  - clear goals of archiving
  - archive policy
  - identification of technical & legal & organizational influence factors
  - monitoring of storage resources
  - consistent indexing of documents
  - superordinated document management

- regular revision of archiving process
- regular usage of archive system
- regular purification of archived data
- regular purification of encrypted data, digital signatures
- regular renewal of technical archive system components
- administrator and user training
- protection of integrity of index databases
- logging of archive access
- regular functionality and recovery tests
- evitation of unsafe data formats
- operation of USB storage media
- regular data backup of system and archive data

Teleworking:

- disposal of data carrier and documents at the work place at home
- missing or insufficient training of teleworkers
- temporal limited availability of teleworkers
- insufficient integration of teleworkers into information flow -> countermeasure: defined information flow between teleworker and institution
- unauthorized private use of company telework computer
- increased probability of theft at home office
- access by family members or visitors

- **countermeasures** for teleworking vulnerabilities:
  - rules for teleworking
  - appropriate setup of home office
  - appropriate storage of business document and data carriers
  - rules for transport of documents and data carriers between home office and company
  - concept for helpdesk and maintenance of home office
  - rules for usage of communication and access possibilities
  - security training for teleworkers
  - rules of deputyship for teleworkers
  - backup strategy and storage of backup data carriers

Outsourcing:

- outsourcing strategy
- contractual regulations
- contractual regulations about the end of outsourcing
- dependence on outsourcing supplier
- spoiling of work atmosphere by outsourcing plans
- technical connection of outsourcing supplier, e.g. usage of insecure protocols in public networks
- incident precaution concept
- dependence on the systems of an outsourcing service provider
- disclosure of data to third parties by outsourcing provider

- **countermeasures** for outsourcing vulnerabilities:
  - outsourcing strategy
  - definition of quality requirements for outsourcing projects
  - appropriate choice of outsourcing supplier
  - contractual arrangements
  - IT security concept
  - emergency plan
  - secure migration
  - planning and maintenance of IT security during outsourcing

Sources: [AER02], [BSI04]

Software engineering process:

- program faults (documented software errors)
  - module internal faults (e.g. syntax, inconsistencies, logic faults, programming fault like program rule violation)
  - module interface faults (interactions with other system components, such as transfer of data or control, data mismatch such as name faults, structural faults, value faults, procedural faults)
  - module functional faults (operating faults: omission or unnecessary operations; conditional faults: incorrect condition or limit values; behavioral faults: incorrect behaviour, not conforming to requirements)
- human errors (cause of program faults)
  - coding or editing errors
  - communication errors within a team (misunderstanding S/W interface specifications)
  - communication errors between teams (misunderstanding H/W interfaces specifications or other team´s S/W specifications)
  - errors in recognizing requirements (misunderstanding specifications or problem domain)
  - errors in deploying requiremenets (misunderstandings, problems implementing or translating requirements into design)
- process flaws (flaws in control of system complexity + inadequacies in communication or development methods)
  - inadequate code inspection and testing methods
  - inadequate interface specifications + inadequate communication (among S/W developers)
  - inadequate interface specifications + inadequate communication (among H/W developers)
  - requirements not identified or understood + incomplete documentation
  - requirements not identified or understood + incomplete design
  - design principle flaws (flaws related to fundamental principles that designers or programmers must follow in order to define proper and understandable interface of functional structures, like inappropriate interface definitions prone to be misunderstood and misdeveloped (definitions inconsistent and distributes, complicated correspondence between definitions, insufficient discrimination between defined items, ambiguous labels defining items)
  - design management flaws (flaws related to the methods and procedures facilitating design management, i.e. how to document and communicate

information on the interfaces and functional structures so that designers and programmers can utilize them properly and evaluate their correctness)
    - lack of methods for recording and referring software interface definitions (inappropriate communication of module calling information, inappropriate communication of global variable or file access information)
    - lack of communication methods between software engineers and hardware engineers (inappropriate communication of hardware physical configuration, inappropriate communication of hardware access information)

- **countermeasures** for software engineering process vulnerabilities:
    - focus on the interface between the software and the system in analyzing the problem domain
    - identify safety-critical hazards early in the requirements analysis
    - use formal specification techniques in addition to natural-language software requirements specifications
    - promote informal communication among teams
    - as requirements evolve, communicate the changes to the development and test teams
    - include requirements for "defensive design"

Source: [Lut93], [NK91]

Further countermeasures which are not specific to any single asset or vulnerability:
Immunity, Integrity, survivability, physical protection, security auditing [Fir03c], virus filter, access authorization (authentication, access rule validation, identification), auditing, alarm, encryption, rapid posting, perturbation (noise addition) [CNYM00], p.205f; standard compliance [DPB+04] p.96; Audit, accountability, controlled Object Reuse, Accuracy, Reliability of Service, Data Exchange [BSI91]; Identification and Authentication (authentication by possession, by knowledge, by characteristic features), Administration of Rights, Verification of Rights, audit, controlled object reuse, error recovery, data communication security, peer entity authentication, access control, data origin authentication, non-repudiation, source code inspection [BSI89]; security audit (=audit of security activities; contains automatic response, data generation, analysis, review, event selection, event storage), security audit review (= tool assisting in review of audit data), assuring the identity of a party participating in a data exchange (proof of origin, proof of receipt), cryptographic functionality (to help satisfy several high-level security objectives; these include (but are not limited to): identification and authentication, non-repudiation, trusted path, trusted channel and data separation), access control, information flow control, residual information protection, rollback (=undo of last operations and return to defined state), stored data integrity, export to outside control, import from outside control, data authentication, flow control, identification and authentication (contains: authentication failures, user attribute definition, specification of secrets, user authentication, user identification, user-subject binding, security management (security attributes, data and functions, revocation, security attribute expiration, security management roles), privacy (includes: anonymity, pseudonymity, unlinkability of several uses of resources, unobservability, abstract machine test, fail secure, physical protection, trusted recovery, replay detection, reference mediation, domain separation, state synchrony protocol, time stamps, data replication consistency, self test, detection and notification of attack, resistance to attack, fault tolerance, priority of service, resource allocation (e.g. quotas, then

no monopolisation of resources possible), limitation on scope of selectable attributes, limitation on multiple concurrent sessions, session locking, access banners, access history, session establishment, trusted path/ channels, explicitely stated security requirements, administrator guidance documents, user guidance, development tools, compliance with implementation standard [CC99]

References

[ABD02] Aagedal, J.Ö., den Braber, F., Dimitrakos, T., Gran, B.A., Raptis, D., Stölen, K.: Model-based Risk Assessment to Improve Enterprise Security. Proc. Of the 5th International Enterprise Distributed Object Computing Conference (2002) 51-62

[AER02] Anton, A.I., Earp, J.B., Reese, A.: Analyzing Website Privacy Requirements Using a Privacy Goal Taxonomy, RE´02 (2002) 23-31

[Ale02] Alexander, I.: Initial Industrial Experience of Misuse Cases. Proc. Of IEEE Joint Internat. Requirements Engineering Conf. (2002) 61-68 http://easyweb.easynet.co.uk/~iany/consultancy/misuse_cases/misuse_cases_in_tradeoffs.htm

[Ale02a] Alexander, I.: Modelling the Interplay of Conflicting Goals with Use and Misuse Cases. GBPM´02 (2002) http://www.ibissoft.se/events/gbpm02/FinalVersions/Environment1.pdf

[Ale03] Alexander, I.: Misuse Cases: Use Cases with hostile intent, IEEE Software, Vol. 20 (1) (2003) 58-66

[AK01] Allenby, K., Kelly, T.: Deriving Safety Requirements Using Scenarios. Proceedings of the 5th International Symposium on Requirements Engineering (2001) 228-235

[Asl95] Aslam, T.: A taxonomy of security faults in the Unix operating system. Master´s thesis, Purdue University (1995)

[BAM01] Buglione L., Abran A., Meli R.: How Functional Size Measurement Supports The Balanced Scorecard Framework For ICT, FESMA-DASMA (2001) 259-272.

[BBH03] Breu, R., Burger, K., Hafner, M., Jürjens, J., Popp, G., Wimmel, G., Lotz, V.: Key Issues of a Formally Based Process Model for Security Engineering (2003) http://www4.in.tum.de/~wimmel/papers/BBHJPWL03_ICSSEA.pdf

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

[BH04] Blakley, B., Heath, C., and members of The Open Group Security Forum: Security Design Patterns, http://www.opengroup.org/onlinepubs/9299969899/toc.pdf

[Bre05] Breu, R.: Systematischer Entwurf zugriffssicherer Systeme, in: Software-Engineering – Objektorientierte Techniken, Methoden und Prozesse in der Praxis, Oldenbourg Wissenschaftsverlag, München (2005) 141-170

[BSI89] BSI = Bundesamt für Sicherheit in der Informationstechnik = German Ministery for Security in Information Technology: IT security criteria (1989) http://www.bsi.bund.de/zertifiz/itkrit/itgruene.pdf

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

[BSI04] BSI = Bundesamt für Sicherheit in der Informationstechnik = German Ministery for Security in Information Technology: IT-Grundschutzhandbuch 2004, http://www.bsi.bund.de/gshb/deutsch/g/g01.html

[BV00] Beizer, B.: “Bug Taxonomy and Statistics” Appendix, SOFTWARE TESTING TECHNIQUES, second edition, Van Nostrand Reinhold, New York, 1990; version as amended by Otto Vinter, see: http://inet.uni2.dk/~vinter/bugtaxst.doc

[CC99] Computer Security Resource Center (CSRC): Common Criteria, Version 2.1, http://csrc.nist.gov/cc/

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

[DKVP03] Dörr, J., Kerkow, D., von Knethen, A., Paech, B.: Eliciting Effciency Requirements with Use Cases. REFSQ - Workshop on Requirements Engineering for Software Quality (2003)

[DVF93] Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisition. Science of Computer Programming, Vol. 20 (1993) 3-50

[DPB+04] Dörr, J., Punter, T., Bayer, J., Kerkow, D., Kolb, R., Koenig, T., Olsson, T., Trendowicz, A.: Quality Models for Non-functional Requirements. IESE-Report Nr. 010.04/E (2004)

[Egy03] Egyed, A.: A Scenario-Driven Approach to Trace Dependency Analysis. IEEE Transactions on Software Engineering, Feb 2003, Vol. 29(2), p.116-132

[EG04] Egyed, A., Grünbacher, P.: Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help. IEEE Software, Vol. 21 (6), (Nov/Dec 2004) 50-58

[Fir03a] Firesmith, D.G.: Common Concepts Underlying Safety, Security, and Survivability Engineering. Technical Note CMU/SEI-2003-TN-033 (2003) http://www.sei.cmu.edu/pub/documents/03.reports/pdf/03tn033.pdf

[Fir03b] Firesmith, D. G.: Security use cases. Journal of Object Technology, 2(3) (2003) 53-64 http://www.jot.fm/issues/issue_2003_05/column6.pdf

[Fir03c] Firesmith, D.G.: Analyzing and Specifying Reusable Security Requirements., 11th IEEE International Requirements Engineering Conference (RE'2003) Requirements for High Assurance Systems (RHAS) Workshop, Monterey, California, September 2003; http://www.sei.cmu.edu/programs/acquisition-support/publications/firesmith-analyzing.pdf

[FC99] Fowler, M., Scott, K.: UML Distilled. Addison-Wesley, 1999

[Hoc97] Hochmüller, E.: Requirements Classification as a first step to grasp quality requirements. REFSQ´97, Presses universitaires Namur (1997) 133-144

[HL98] Howard, J.D., Longstaff, T.A.: A Common Language for Computer Security Incidents, SANDIA REPORT, SAND98-8667, Sandia National Laboratories, 1998

[Iba98] Ibanez M: Balanced IT Scorecard Generic Model Version 1.0. European Software Institute, Technical Report, ESI-1998-TR-009, May 1998.

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

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

[ISO92] Standard DIN EN ISO 9241, Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten (1992)

[ITU01] ITU – Telecommunication Standardization Sector Draft Specification of the Goal-Oriented Requirements Language (Z.151), September 2001

[KKB+98] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, S.J.: The Architecture Tradeoff Analysis Method. Software Engineering Institute, Technical Report CMU/SEI-98-TR-008 (1998) http://www.sei.cmu.edu/pub/documents/98.reports/pdf/98tr008.pdf

[KKC00] Kazman, R., Klein, M., Clements, P.: ATAM: Method for Architecture Evaluation. CMU/SEI-2000-TR-004, Software Eng. Inst., Carnegie Mellon University (2000) http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr004.pdf

[KMT04] Killourhy, K.S., Maxion, R.A., Tan, K.M.C.: A Defense-Centric Taxonomy Based on Attack Manifestations. International Conference on Dependable Systems & Networks: Florence, Italy (2004) 102-111

[KN92] Kaplan R.S., Norton D.P.: The Balanced Scorecard: Measures That Drive Performance. Harvard Business Review 70(1) (1992) 71-79.

[LBDJ03] van Lamsweerde, A., Brohez, S., De Landtsheer, R., Janssens, D.: From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering. http://www.cs.toronto.edu/~jm/2507S/Readings/avl-RHAS03.pdf

[LBMC94] Landwehr, C.E., Bull, A.R., McDermott, J.P., Choi, W.S.: A taxonomy of computer program security flaws, with examples. ACM Computing Surveys, 26(3) (Sept. 1994) 211-254

[Lut93] Lutz, R.R.: Analysing software requirements errors in safety-critical embedded systems. Proceedings of RE'93 (1993)

[LW98] van Lamsweerde, A., Willemet, L.: Inferring Declarative Requirements Specifications from Operational Scenarios. IEEE Trans. on Software. Engineering, Special Issue on Scenario Management (December 1998) 1089-1114

[LYM03] Liu, L., Yu, E., Mylopoulos, J.: Security and Privacy Requirements Analysis with a Social Setting. Proc. RE´03 - Intl. Conf. On Requirements Engineering, Monterey, California, September (2003)

[MYBM91] Maclean, A., Young, R.M., Belotti, V.M.E., Moran, T.P.: Questions, Options and Criteria: Elements of design space analysis. Human Computer Interaction, 6 (3&4) (1991) 201-250

[MDF99] McDermott, J., Fox, C.: Using Abuse Case Models for Security Requirements Analysis. 15th Annual Computer Security Applications Conference (1999) 55-6

[MEL01] Moore, A.P., Ellison, R.J., Linger, R.C.: Attack Modeling for Information Security and Survivability. Technical Note CMU/SEI-2001-TN-001 (March 2001)

[MRD05] Meyer, N., Rifaut, A., Dubois, E.: Towards a Risk-Based Security Requirements Engineering Framework. REFSQ - Proc. Of Internat. Workshop on Requirements Engineering for Software Quality (2005)

[MHN04] Moffett, J. D., Haley, C. B., Nuseibeh, B.: Core Security Requirements Artefacts. Technical Report No 2004/23, Department of Computing, The Open University, UK (2004) http://computing-reports.open.ac.uk/index.php/2004/200423

[NK91] Nakajo, T., Kume, H.: A Case History Analysis of Software Error Cause-Effect Relationships. IEEE Trans. Software Eng. 17 (8) (1991) 830-838

[OMG04] Object Management Group: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. (2004) http://www.omg.org/docs/ptc/04-09-01.pdf

[PDKV02] Paech, B., Dutoit, A., Kerkow, D., von Knethen, A.: Functional requirements, non-functional requirements and architecture specification cannot be separated - A position paper. REFSQ - Proc. Of Internat. Workshop on Requirements Engineering for Software Quality (2002)

[PX02] Pauli, J., Xu, D.: Misuse Case-based Analysis of Secure Software Architecture. (2002) http://cs.ndsu.edu/~dxu/research/NDSU-CS-TR-04-XU02.pdf

[Ran97] Ranum, M.J.: A taxonomy of Internet attacks. Slide Presentation (1997), available on the Internet at http://pubweb.nfr.net/~mjr/pubs/pdf/internet-attacks.pdf

[Ric03] Richardson, R.: 2003 CSI/FBI Computer Crime and Security Survey. Computer Security Institute. (2003) http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2003.pdf

[RW05] Regev, G., Wegmann, A.: "Where do Goals Come from: the Underlying Principles of Goal-Oriented Requirements Engineering", Proc. RE05 (13th Int´l Requirements Engineering Conference), La Sorbonne, France, Aug29-Sept2, 2005, pp.353-362

[SO00] Sindre, G., Opdahl, A.L.: Eliciting Security Requirements by Misuse Cases. Proceedings of TOOLS Pacific 2000 (2000) 120-131

[SO01] Sindre, G., Opdahl, A.L.: Templates for Misuse Case Description. REFSQ - Proc. Of Internat. Workshop on Requirements Engineering for Software Quality (2001) 125-136

[SM98] Sutcliffe, A., Minocha, S.: Scenario-based Analysis of Non-Functional Requirements. REFSQ - Workshop on Requirements Engineering for Software Quality (1998) 219-234

[Wie02] Wiegers, K.E.: Success Criteria Breed Success, The Rational Edge, 2(2), 2002

Case study in section 6 was removed because it is confidential

[Xie04] Xie, N., et al.: SQUARE Project: Cost/Benefit Analysis Framework for Information Security Improvement Projects in Small Companies. Technical Note CMU/SEI-2004-TN-045 (2004)