Quality Misuse

Andrea Herrmann and Barbara Paech

University of Heidelberg, Faculty of Mathematics and Computer Science, Software Engineering Group, 69042 Heidelberg, Germany
{herrmann, paech}@informatik.uni-heidelberg.de
http://www-swe.informatik.uni-heidelberg.de/

Abstract. There are several methods for the derivation and analysis of detailed non-functional requirements. They often are designed for a special application, like 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, efficiency) to develop a misuse-based method for deriving detailed non-functional and functional requirements. Doing so, we find that generalizations must be made to the definitions of the misuse case concepts, and new concepts must be included. With this paper, we want to stimulate a discussion about the general usefulness of our approach and definitions.

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 [13, 14] for detailing the quality attribute "security", or ATAM (Architecture Trade-off Analysis Method) [8] for evaluating given architectural alternatives. We follow a suggestion of I. Alexander [1]: "There is scope for further work applying Misuse Cases to elicit Usability requirements." In [2] he himself applies them to reliability, maintainability and portability. We go a step further and apply misuse cases also to all other quality attributes (QAs), what has not been done explicitly and as systematically so far. 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. Both, intended system functionality and threats to it, in our view are essential to understand quality. Furthermore, misuses help to relate NFR to functional requirements (FR), which in our opinion must be considered "in a tightly integrated approach" [12]. Sutcliffe and Minocha [15] suppose that NFR express constraints on FRs and design.

Our aim is to develop a misuse-based method for deriving detailed non-functional and functional requirements. In a first step, we here define the necessary concepts. Doing so, we found that a more general terminology is necessary, because misuse cases are tailored to security requirements, and new concepts must be included. With this paper, we want to stimulate a discussion about the general usefulness of this approach and our definitions.

This article is structured as follows: Section 2 presents our generalized terminology. In section 3, we more specifically treat several software QAs. In section 4, we discuss our results and describe perspectives for further research.

2 Generalized Misuse Case Terminology

We chose the misuse case approach as a basis for defining NFRs ([13, 14]. Misuse cases typically describe a misuser´s behaviour. From our sources, we extracted the one general principle behind the misuse cases and several other methodologies (to evaluate NFRs [15] or architecture [3, 4, 8], but also in security [9, 10, 11]). It is: An asset is to be protected from a threat or misuse by defining countermeasures. We adopt concepts from the above methods where necessary. See figure 1 for an overview of our terminology.

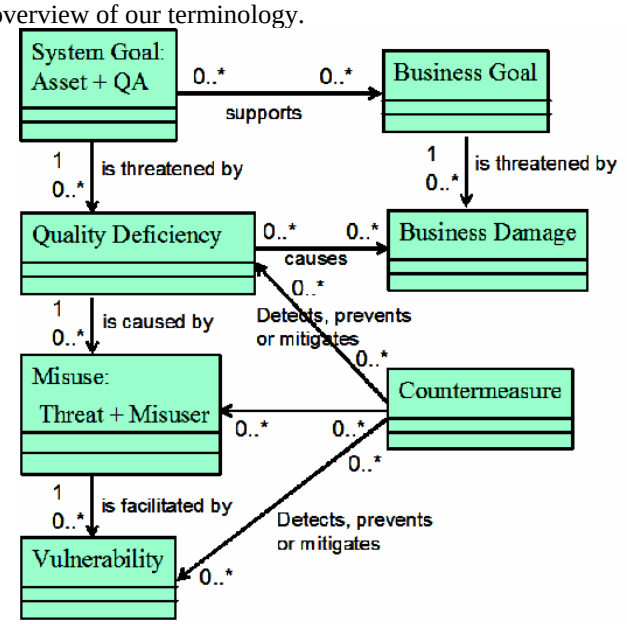

Fig. 1. our concepts: see text of section 2 for definitions and explanation

An *asset* is anything which must be protected. The misuse case literature expects an "asset" to be "data, communications, services, hardware components, and personnel" [5]. ATAM [8] mainly considers architecture components. For our purpose, assets can be all of these ("services" might be use cases). Furthermore, it is necessary to combine the asset with a QA or NFR to specify which aspect of the asset

needs protection. Therefore, we always consider the combination of an asset plus its QA. We call this pair a *system goal*. To identify use cases which might be assets, not only normal use (these involve typical uses of the existing system) is relevant, but also *growth scenarios* (these cover anticipated changes to the system; relevant for maintainability, interoperability and portability) and *exploratory scenarios* (these cover extreme changes that are expected to "stress" the system; relevant for security, reliability, efficiency, recoverability) [8].

The reason why the system 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 the *quality deficiency*. It is not necessarily the exact opposite of the QA. For example, if the QA is "availability", the quality deficiency can consist in temporary not-availability for all users or for certain users, definite destruction of the data, manipulation of the data, and many more.

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

A *threat* is an action which causes the quality deficiency. So it actively threatens the asset + QA. It is the. Firesmith lists the security threats "theft, vandalism, fraud, unauthorized disclosure, destruction, extortion, espionage, trespass" [Fir].

The threat is usually performed by a *misuser*. This could be a person (hacker), other systems or forces of nature like fire and thunderstorm. Security is the only QA where the human *misuser* either is an intruder or a user who executes a use case s/he 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. All other QAs are threatened by regular users or other systems 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 [8]), for example for portability and maintainability.

Often, the threat is facilitated or even provoked by a *vulnerability*. A vulnerability is a property of the system and might be a code flaw or a design flaw, but also any – even wanted – property of the system, if it can be misused with respect to the asset and QA. 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. 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 asset and QA to be protected, to decide whether there is a potential misuser who might try to threaten the asset´s NFR.

A *misuse* is described by a misuse scenario, including misuser, vulnerability, threat and its consequences. The misuse is written 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 in the form of 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 threat is described by a scenario and the misuse consequences (quality deficiency) is the post-condition of the misuse

case. The vulnerability might be a pre-condition of the misuse case if it is a prerequisite for the threat to take place.

To handle threats and misuses, we need *countermeasures*. We adopt this term from Sutcliffe and Minocha [15]. Countermeasures either detect, prevent or mitigate the threat, the system vulnerability or the quality deficiency having the predicted consequences [14, 6]. The countermeasures are use cases, new or extended exception scenarios of use cases, use case NFRs (including metrics), architectural or other constraints (see: [3, 4, 8, 13, 15]). They complete the system requirements and also the requirements on the development, project management and maintenance process.

3 First Results for Different Quality Attributes

During our analysis, we found fundamental differences among the various QAs in terms of misusers and assets. Therefore, we regarded the ISO 9126 [7] QAs separately. In some cases, we treat quality attributes of the top level, sometimes sub-attributes, e.g. for functionality, which is a collection of several different QAs like security. Below we summarize our results of analyzing QAs in terms of misusers, threats and countermeasures. The results are not intended to be complete, but rather meant to test the usefulness of our concepts. Although we analysed all QAs, for the sake of shortness, we here omit the results for safety, suitability, accuracy and interoperability. We will not name business goals and business damages as they depend strongly on the application domain.

QA: Security
Assets: data, communications, services (=use cases), hardware components, personnel (see [5])
Quality Deficiency: loss of availability, integrity (completeness and accuracy), privacy/ confidentially, and operational security [3], immunity, survivability [5]
Threats: theft, vandalism, fraud, unauthorized disclosure, destruction, extortion, espionage, trespass [5]; **Misusers:** hackers, crackers, disgruntled employees, international cyber-terrorists, industrial spies, governmental spies, foreign military, etc. [5]
Vulnerabilities: misuse was not foreseen by the system design; carelessness like poorly protecting a password
Countermeasures: Identification, authentification, authorization, intrusion detection, nonrepudiation, privacy/ confidentiality, immunity, integrity, survivability, physical protection, security auditing [5]; virus filter, access authorization (authentication, access rule validation, identification), auditing, alarm, encryption, rapid posting, perturbation (noise addition) [3], p.205f; standard compliance [4]

QA: Reliability in terms of Maturity and Fault tolerance
Assets: data, communications, services, hardware components, personnel (see [5]), use cases
Quality Deficiency: system breakdown by too many users at a time, breakdown by user error
Threats: user error, heavy usage; **Misusers:** one or several users
Vulnerabilities: user error/ load was not foreseen by the system design; single point of failure
Countermeasures: maturity, fault tolerance, formal methods, redundancy, modularisation, catching exceptions, threads, standard compliance [4], p.40; fault detection, recovery, failover, monitoring, sanity check, [4], p.50ff

QA: Reliability in terms of Recoverability
Assets: use case "recover system after breakdown"
Quality Deficiency/ Threats: no backup, backup restorage does not work, backup restorage takes long; **Misusers:** user, administrator; **Vulnerabilities:** system is not recoverable or only with high effort; insufficient training, unreliable work style of administrator;
Countermeasures: Use cases "backup" and "backup restoring"; training, automated backup facility

QA: Usability (= understandability, learnability, operability)
Assets: use cases and their correct accomplishment
Quality Deficiency: 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 to use the system, inefficiency of user tasks, non-adherence to processes, incomplete accomplishment of a use case, no mental model of system
Threats: errors in the course of normal use, impatience, missing understanding of how to use the system correctly; **Misusers:** end-user, administrator, maintainer
Vulnerabilities: bad user interface, too little documentation, insufficient training
Countermeasures: one can build upon rules and standards like ISO 9241; e.g. constraints on the interface definition, use case scenarios; [4] p.75 documentation, feedback, metaphors

QA: Time efficiency
Assets: duration of a use case, but also system components or communication; normal use scenarios and also exploratory scenarios
Quality Deficiency: inefficient use, slowdown or breakdown; Threats: high load
Misusers: end-user, maintainer and administrator, **Vulnerabilities:** bottleneck; usage of a component by too many users or requests; rules like obligation of all users or many users to do the same task at the same time, e.g. produce the same individual report on Friday afternoon at 4 pm or to log in by work or shift start immediately
Countermeasures: Constraints on system use, constraints on hardware or software; [4] p.62: observation of arrival patterns, observation of execution time, locality, parallelism, spare schedule; locality, parallelism, caching, sharing [4] p.68

QA: Resource efficiency
Assets: resource (hardware), bandwidth; processor, network, memory, secondary storage [4] p.62
Quality Deficiency: overload of hardware, slowdown or breakdown, memory overflow
Threats: high load, denial-of-service attack
Misusers: end-user or end-user groups, maintainer and administrator, attacker;
Vulnerabilities: inadequate resources; rules (like for "time efficiency")
Countermeasures: Constraints on system use, constraints on workload distribution, capacity of memory/ network/ processor/ etc, type and position of devices [4]; security

QA: Maintainability, i.e. analysability, changeability, stability and testability
Assets: status quo of all use cases supported by the system so far, also maintenance/ growth use cases and their quality (in terms of functionality, reliability and efficiency)
Quality Deficiency: too high costs for maintainance; loss of system quality with changes
Threats: error, change without former impact analysis, error during impact analysis, (anticipated) changes to the system; **Misusers:** Maintainer, **Vulnerabilities:** badly analysable code, badly changeable code, instable code, badly testable code
Countermeasures: suitable constraints on the maintenance process (instructions demanding that and how impact analysis has to be done)**;** constraints on the software development process

(e.g. coding guidelines for assuring analysability, changeability, stability and testability); see [4], p.17 and 25: encapsulation/ modularity, structuredness, reduction of complexity, tracing, documentation, reuse, automatic test case generation, testsuite interface generation, separation, indirection, abstraction, location transparency, layered architecture, moduls model + view + controller, publish-subscribe, traceability matrix, interface consistency

QA: Portability
Assets: growth scenarios: adapt the system, install on other system (hardware, operating system, etc.; transfer and configure), replace part of the system by another (see ISO 9126 [7]); their protectable QAs are not only portability, but also functionality, reliability and efficiency
Quality Deficiency: new configuration does not work any longer (functionality), data are flawed (reliability), communication is slowed down (efficiency)
Threats: error, **Misusers:** administrator,
Vulnerabilities: system not portable or only with high effort
Countermeasures: administrator training, developer training; adherence to standard, independence, modularity, operating system functionality [4]

4 Summary

This position paper outlines concepts for an analysis of NFRs of a software system. The approach used here is based on the concept of misuse cases and is illustrated by applying it to the QAs defined by ISO 9126. We believe that such an analysis of NFRs in terms of assets, threats, misusers and countermeasures helps to complement software and project requirements. To support a complete view on system quality, we consider not only end-users, but also system administrators, maintainers and intruders to the system. We take account of use, growth and exploratory scenarios. We believe an important value of our approach is its general applicability to all QA´s and the integration of NFR and FR. It is clear enough to be used for communication with customers and end-users.

In a first case study (not described here) the concepts worked well and we did not come across any non-functional requirements which could not be specified in this form.

Our actual lists of assets, threats and countermeasures are not meant to be complete. In future work we plan to follow Firesmiths [5] proposal to compile reusable checklists by gathering content from diverse literature sources.

This work is seen as an intermediate result to discuss our idea. Our long term goal is to not only integrate terminologies – as we did here – but also methods. For now, we did not give any method or direction on how to derive requirements, but only illustrated the concepts and their relationships. We also intend to integrate information about probabilities of misuses and severity of damage, as well as efficiency of countermeasures, and to consider relationships between and among threats and countermeasures [6, 14]. Furthermore, we want to treat conflict negotiation among NFRs and countermeasures systematically.

References

1. Alexander, I.: Initial Industrial Experience of Misuse Cases. Proc. Of IEEE Joint Internat. Requirements Engineering Conf. (2002) 61-68
2. Alexander, I.: Misuse Cases: Use Cases with hostile intent, IEEE Software, 20 (1) (2003) 58-66
3. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Acadamic Publishers (2000)
4. 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)
5. Firesmith, D. G., Analyzing and Specifying Reusable Security Requirements. http://www.sei.cmu.edu/programs/acquisition-support/publications/firesmith-analyzing.pdf
6. Firesmith, D. G.: Security use cases. Journal of Object Technology, 2(3) (2003) 53-64
7. International Standard ISO/IEC 9126. Information technology -- Software product evaluation -- Quality characteristics and guidelines for their use
8. Kazman, R., Klein, M., Clements, P.: ATAM: Method for Architecture Evaluation. CMU/SEI-2000-TR-004, Software Eng. Inst., Carnegie Mellon University (2000)
9. Lamsweerde, A. van, Brohez, S., Landtsheer, R. De, 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
10. Lamsweerde, A. van, Willemet, L.: Inferring Declarative Requirements Specifications from Operational Scenarios, IEEE Trans. on Sofware. Engineering, Special Issue on Scenario Management, December 1998, 1089-1114
11. Liu, L., Yu, E., Mylopoulos, J.: Security and Privacy Requirements Analysis with a Social Setting, Proc. RE´03 - Intl. Conf. On Requirements Engineering, September, 2003
12. Paech, B., Dutoit, A., Kerkow, D., von Knethen, A.: Functional requirements, non-functional requirements and architecture specification cannot be separated. REFSQ - Internat. Workshop on Requirements Engineering for Software Quality (2002)
13. Sindre, G., Opdahl, A. L.: Eliciting Security Requirements by Misuse Cases. Proceedings of TOOLS Pacific 2000 (2000) 120-131
14. Sindre, G., Opdahl, A. L.: Templates for Misuse Case Description. REFSQ - Internat. Workshop on Requirements Engineering for Software Quality (2001) 125-136
15. Sutcliffe, A., Minocha, S.: Scenario-based Analysis of Non-Functional Requirements. REFSQ - Workshop on Requirements Engineering for Software Quality (1998) 219-234