MAGISTERARBEIT

Titel der Magisterarbeit

Functional Security Testing

- An approach for a global Telecommunication company -

Verfasser

Clemens Kögler, Bakk.

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2009

Studienkennzahl lt. Studienblatt: A 066 926
Studienrichtung lt. Studienblatt: Wirtschaftsinformatik
Betreuer: Univ.-Prof. DDr. Gerald Quirchmayr

Acknowledgements

As my master thesis comes to an end now, I would like to thank all those who supported me during its creation.

Special thanks go to my supervisor at *TeliaSonera*, Dr. Albin Zuccato, who provided me with ideas for this work and has supported me with indispensable information and feedback during it's creation. Additionally I would also like to thank *TeliaSonera* and especially the *Product Security* group, which created an open minded environment that enabled me to broaden my view on different aspects of information security. My gratitude at *University of Vienna* goes to Prof. Gerald Quirchmayr, who initially made this thesis happen. My Thanks go also to the Erasmus [EU] office at University of Vienna and the Department of Computer and Systems Sciences (DSV) [SU] at Stockholm University.

Further I want to thank my parents Michael and Helene Kögler, who made it possible for me to go to university and provided invaluable support during my years of study. Last but not least my thanks go to Nicole Filip, who showed a lot of appreciation and encouraged me during the creation of this work.

Abstract

Functional Security Testing can be used to assure that the security functionality of a system works as specified. Thereby the focus lies not on a malicious breakup of a system, but on achieving assurance that its specified functionality can be trusted. In this work we present an approach for organizing and conducting functional security tests. It can be used to perform such tests for various systems and security requirements. Furthermore, it is highly flexible and extendable, which allows it to be adapted to the specific needs of an organization and its systems. To show the basic practicability of the theoretical concepts we present, rudimentary test data are created for the developed approach. The test data is necessary to make the theoretical concepts feasible in practice. The approach includes a structure which provides guidance for its enrichment through the development of quantitative test data. This is considered as a necessity due to the criticality of this task. To evaluate the correctness and practical applicability of the approach a validation is shown. The presented validation is threefold. At the beginning, we validate the theoretical concepts in respect to there common criteria [ISO07] conformance and compliance to other relevant standards and best practices. Subsequently, we indicate the completeness of the test cases by performing a coverage assessment. Finally, we validate the applicability of the approach in a real-life project.

Summarized it can be said that this work provides a holistic treatment of functional security testing and provides a flexible and extendable approach for performing and organizing functional security tests.

Contents

1 Motivation 1
- 1.1 The research question 2
- 1.2 TeliaSonera 2
- 1.3 Layout of this work 4
- 1.4 Methodology 5

2 Software and System Testing 7
- 2.1 Standards 7
  - 2.1.1 ISO/IEC 15408 - Common Criteria 8
  - 2.1.2 ISO/IEC 27002 - Code of Practice for Information Security Management 9
  - 2.1.3 ISO/IEC 27001 - Specification for an Information Security Management System 10
  - 2.1.4 IEEE Std 829-1998 - Software Test Documentation . . 11
  - 2.1.5 ISO/IEC 21827 - Systems Security Engineering - Capability Maturity Model 11
  - 2.1.6 ISO/IEC 20000 - IT Infrastructure Library 12
- 2.2 Verification concepts 14
- 2.3 The Software Development Life Cycle 16
- 2.4 The Secure Development Life Cycle 19
- 2.5 Testing 22
  - 2.5.1 Scope of testing 25
  - 2.5.2 Depth of testing 26
  - 2.5.3 The software testing process 27

3 System Security Testing 29
- 3.1 Security- and Traditional Testing 29
- 3.2 Penetration Testing 31
- 3.3 Functional Security Testing 33

3.3.1 Functional Security Testing Techniques 35
3.3.2 Definition of Functional Security Testing 36
3.4 Regression Testing in Security 37

4 The Functional Security Testing Approach 39
4.1 TeliaSonera's Security Mechanism Reference Table 40
4.2 The process for Functional Security Testing 42
4.3 The structure of a Functional Security Testcase 46
4.4 A database for Functional Security Testing 47
4.4.1 Identification of entities 48
4.4.2 Top level schema 49
4.4.3 Relational schema 52
4.4.4 Implementation 57
4.5 Creating content for the functional security testing approach 59

5 Derivation of Functional Security Test Cases 63
5.1 Predefined Structure 63
5.1.1 Regression Levels 65
5.2 Crafting of Functional Security Test Cases 65
5.2.1 Example A - Standard One Factor Authentication . . . 67
5.2.2 Example B - SSL/TLS 75

6 Validation of the Functional Security Testing Approach 85
6.1 Validation against the Common Criteria 85
6.1.1 ATE_COV: Coverage 86
6.1.2 ATE_DPT: Depth 87
6.1.3 ATE_FUN: Functional tests 88
6.2 Validation against other relevant parts of standards 90
6.2.1 Validation against ITIL 91
6.3 Validation of requirements coverage 92
6.4 Validation in a real-life project 96
6.4.1 Design test plan 97
6.4.2 Run program with test data 98
6.4.3 Compare results to test cases 98
6.4.4 Analysis 99

7 Conclusion 101
7.1 Contributions of this work 102
7.2 Outlook and further research 103
7.2.1 Attack Patterns 103
7.2.2 Software Weakness Collection 104
7.2.3 More Automatization 105

7.2.4 Compliance to standards 105
References 107

Functional Security Testing Approach Database Schema 115

Functional Security Test Cases 121
B.1 CA0 - Authentication 121
B.1.1 ME0 - One factor 121
B.1.2 ME1 - One factor 121
B.1.3 ME2 - One factor strong authentication 122
B.1.4 ME3 - Two factor strong authentication without a seperate physical token 123
B.1.5 ME4 - Two factor strong authentication with seperate physical token 124
B.2 Further categories 124

Chapter 1
Motivation

Information systems are an integrated and indispensable component of almost every organization. Consequently, the security of these systems and as a result, the security of the whole business have become an important factor. Information (in)security has become so important that it can, in an extreme case, put an organization out of business. This can happen when the security does not meet the given legal requirements, or when the trust of the business partners is lost due to a security breach. Both can be affected by software security related issues, which are further described in [McG06]. That is why there must be clearly defined security requirements and why they must be fulfilled by an information system. Especially functional security requirements, detailing the non-functional requirements (see chapter 3) towards security from system engineering, play a major role when it comes to security assurance. Hence the security expectations of a system must be codified through security requirements. These security requirements have to be functionally tested to show the system's compliance to them.

Functional testing cannot be used to show the overall security of the whole system, as testing per se does not allow this for large systems [Dij70]. However, it can be used to get an indication that the functional requirements are met. In contrast to penetration testing, functional testing is not utilized to compromise or break a system, but rather to show that the security safeguards adhere to the functional security requirements. The fulfillment of functional security requirements is the area on which this thesis is focused.

The target audience this thesis aims at are mainly software- and security engineers, but it can also be valuable for more business-oriented persons, which want or have to know more about how to improve the security of software. As this thesis is written in cooperation with an internationally operating company, it has strong relevance to problems and threats global organizations face. However, it is based on academic pillars which are used

to assure that this work is not only beneficial for the cooperating company, but still provides valuable information for the scientific community.

1.1 The research question

The research question, which we investigate in this work aims at functional security tests and their utilization for testing security requirements.

In this thesis we will show that *it is possible to create a correlation between functional security tests and given security requirements, which enables a direct mapping of test cases to the underlying requirements*.

In course of this thesis we create an approach for functional security testing based on predefined test cases, which enables a direct correlation to security requirements. Hence we construct functional security test cases which can be used to verify and validate the functional correctness of the linked requirement(s). The transition from requirements to related test cases should be made semi-automatically. The manual execution of the test cases itself is assigned to the tester, as fully-automatic tests are outside the scope of this work.

To measure the quality of the approach, parts of the *Common Criteria*, as defined in [ISO07], are taken as reference. My goal is that the approach enables systems to fulfill the requirements for an EAL2 classification within the testing part (Class ATE: Tests). The requirements for EAL2 are shown in figure 1.1 by the highlighted areas. The ATE_IND family is defined as irrelevant in this case as it refers to the evaluator and not the developer. We discuss this in more detail in section 6.1.

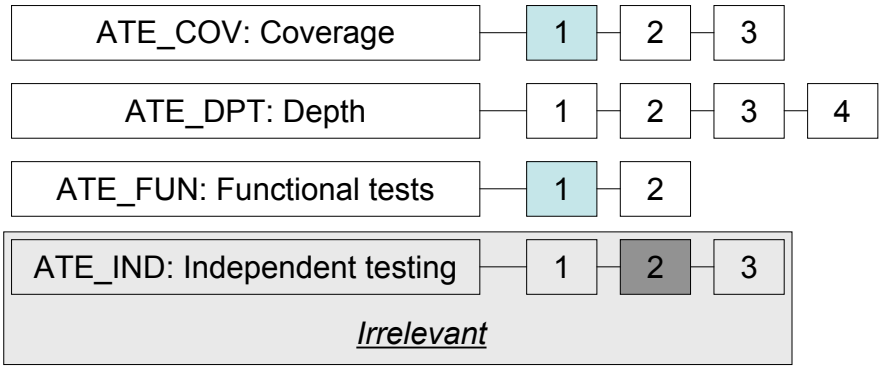

Fig. 1.1 Common Criteria, Class ATE: Tests, adapted from [ISO07]

A further measure is the coverage of the requirements by there test cases. The developed test data should reach a sophistication level that provides a holistic coverage of the linked requirements.

As the practical usage of the approach is of special interest for *TeliaSonera*, the applicability of the approach should be assessed and measured in a real project.

1.2 TeliaSonera

This thesis is a cooperation between *University of Vienna* and *TeliaSonera*. Hence this work is shaped around processes and structures of *TeliaSonera*. This results in the thesis being able to test academic ideas against real-world requirements. Due to the opportunities given by *TeliaSonera*, this work is not only limited to commonly accepted ideas and theoretical investigations, but can also build on the latest practical knowledge provided by the researchers of the company.

As stated in [Tel08], *TeliaSonera* considers itself as a global player in telecommunication, which gives this thesis the european outlook my erasmus exchange stay at DSV is aimed at.

> TeliaSonera is the leading telecommunications company in the Nordic and Baltic region and also holds strong positions internationally in mobile communications in Eurasia, including Turkey and Russia. At the end of 2006 TeliaSonera successfully launched mobile services in Spain. We offer reliable, innovative and easy-to-use services for the transmission and packaging of sound, images, data, information, transactions and entertainment. [Tel08]

1.3 Layout of this work

This thesis is divided into four parts. The first (chapter 2 and 3) contains theoretical foundations of testing with a focus on security. This base is used by the subsequent chapters (4 and 5) to develop an approach which satisfies the research question of this thesis. The third part, presented in chapter 6, validates the quality of the developed functional security testing approach. Finally, chapter 7 summarizes the contributions of this work and presents ideas for further research in this area. Each of the chapters is now described in more detail to give an overview of its content.

Part 1 - Theoretical Fundamentals

In chapter 2 we give an overview of standards and best practices which influence this work. Further we discuss different testing concepts and observe them according to their suitability for this thesis. Subsequently we give a

short overview of a software development process. As testing is of special interest for the further work the focus lies on how testing is related to this process and its steps. Further the *Software Development Process Lifecycle* and the *Secure Development Lifecycle* are shortly described to show which role testing can play in these development processes. Subsequently an introduction to testing is given which contains a description of common testing techniques.

The next chapter (3) aims especially on security testing in contrast to testing in general. We explain why security testing is different from traditional testing and the resulting implications. Selected security testing techniques and their differences are described. As an important cornerstone for the further work a definition of functional security testing is given in this chapter. We recommend this chapter, or especially the section about functional security testing, to the reader as this definition forms an important pillar for the rest of the work.

Part 2 - Development of the Functional Security Testing Approach
In contrast to the preparatory chapters before, this part forms the core of the thesis. Chapter 4 shows the development of all static and dynamic aspects of the functional security testing approach. It starts with the gathering of requirements, describes the derived top level schema and leads subsequently to executable source-code. This represents the design part of the approach and the skeleton for the further work. To attach flesh for the developed skeleton, content for the approach is created. We call this quantitative enrichment, which is described in chapter 5. As this should be continuously done during the later usage in real-life, a process is described in chapter 4, which structures the necessary activities and artifacts for adding new content to the approach.

Part 3 - Validation of the Functional Security Testing Approach
In this part, which consists of chapter 6, the developed functional security testing approach is validated against various criteria. First, the qualitative aspect of the approach is examined by comparing it with prerequisites from the *Common Criteria* [ISO07] and other relevant standards. Subsequently the quantitative component is validated. First we discuss various methods and finally select one of them to perform the validation. The last observed aspect of the approach is the practical applicability for *TeliaSonera*. This part is of special interest for substantiating the value of this work. Hence it provides indispensable information about its validity.

Part 4 - Conclusion and Outlook
The last part summarizes the contributions of this thesis to the scientific community. Further an outlook is given for future research in this area. This outlook includes not only general further research topics connected to this work, but also concrete ideas of how the developed approach could be ex-

tended. We recommend this chapter to every reader who is interested in the scientific value of this thesis or in the further extension of the developed approach.

Appendix

The appendix consists of two parts. Appendix A contains the complete source code for the static structure of the approach from chapter 4. In appendix B an overview of the created content for the functional security testing approach is given. It has to be mentioned that a full extract of the content can not be provided due to security concerns of *TeliaSonera*.

1.4 Methodology

The starting point for this work is the assumption that *it is possible to create a correlation between functional security tests and given security requirements, which enables a direct mapping of test cases to their underlying requirements*. According to that we chose a deductive methodology for creating this thesis. This enables me to start from a holistic top-level view and then further dig deeper into the more specific problem areas. Figure 1.2 shows the creation process of this work. After an initial phase of information gathering about testing with a special focus on security testing and relevant standards, we compiled the theoretical foundations of this work. This is represented by chapter 2 and 3. Subsequently the requirements for a functional security testing approach have been collected. Thereby the focus was to integrate the approach into given structures and methods of *TeliaSonera*. Further international standards had an impact on this task, but as some of them are already integrated into *TeliaSonera*, it is nearly impossible to distinguish between the impact of environmental factors defined by *TeliaSonera* and the influence of common standards and best practices. However, the requirements have been implemented and the functional security testing approach has been created in chapter 4. This part represents the qualitative aspects of this thesis. According to the developed approach we created quantitative data in chapter 5. It has to be mentioned that in this part not only *TeliaSonera's* environmental factors played a major role, but also expert knowledge. Subsequently we evaluated and validated the created qualitative and quantitative parts in chapter 6. Finally a conclusion has been written. This includes not only a summary of what has been done, but also related research topics and ideas for further improving the approach in respect to *TeliaSonera's* business needs and compliance to international standards.

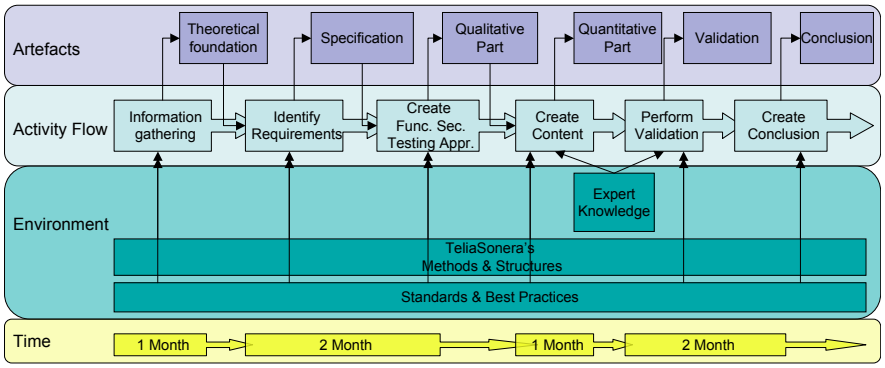

Fig. 1.2 Master Thesis Methodology

Chapter 2
Software and System Testing

In this chapter we give an overview of software and system testing. First we discuss some standards and best practices which influence the development of the functional security testing approach. Subsequently we describe different concepts of testing methods and discuss their suitability for the given requirements. After these foundations for the thesis are explained, we give an introduction into security testing and its relation to the development process of a software system. First, a generic example of a simple and flexible development process for software is presented. Due to the aim of this work the focus lies on security issues and especially testing in this process. This is followed by a description of methods and functions which can be included in the development process activities to increase the security of the developed system. These two parts give an overview where in the development process security issues have to be taken into account and which methods can be used to increase the security level of the system. The testing part, which starts with section 2.5, gives an introduction to different testing approaches with a focus on their aims and fields of application. In this part we introduce a classification model, which can be utilized to differentiate and organize different testing approaches within a coordinate system. In the last part of this chapter we give a short overview of a testing process.

2.1 Standards

To embed this work into commonly used practices and knowledge, some international standards are taken into account. Subsequently we give a short description of the used standards. Thereby the focus is not on the standards themselves, but on their integration into this work. It has to be mentioned that there are many other international standards available, but in respect

to the environmental preconditions given by *TeliaSonera* and the scope of this work, only the listed ones are used in this thesis. However, there are other commonly used (national) best practices which are worth mentioning. The *National Institute of Standards and Technology (NIST)* [NIS] has published a series of best practices for information security, which is called *NIST Special Publication 800-xx series*. The *Bundesamt für Sicherheit in der Informationstechnik (BSI)* [BSI] has issued the *IT-Grundschutzhandbuch (IT-Baseline Protection Manual)* [GHB04], which is concerned about protecting information on a basic level to build a foundation for further improvements in information security when needed. Both best practices are popular in the security community, but are out of scope for this work.

The subsequently discussed standards have an influence on this thesis. Still, the primary aim of this work is to create an approach which fits to the needs of *TeliaSonera*. Therefore standards are only used if they do not compromise a sound integration. However, a lot of knowledge from a broad range of standards is already inherent in *TeliaSonera's* processes and structures. Hence, even if integration is the main factor which influences the functional security testing approach, this implies also compliance to standards to a certain degree.

2.1.1 ISO/IEC 15408 - Common Criteria

This standard is often referred to as *Common Criteria* [ISO07]. The main purpose of the *Common Criteria* is to provide requirements for defining security functionality of information systems and to provide assurance requirements for this functionality. It is divided into three parts. Part 1 gives an introduction into the structure and aim of the standard. Part 2 lists functional components, which can be used to specify the security functionality of a system. Part 3 lists assurance requirements. These requirements are used to evaluate the correctness of an implemented system from various perspectives. Different *Evaluation Assurance Levels (EAL)* from *EAL1* to *EAL7* are defined. The increasing number of the level refers to an increasing demand put on the assurance requirements of the functional components. Both, the functional components and the assurance requirements can be combined to create a *Protection Profile (PP)*. This profile forms an implementation independent specification of security functionality and assurance levels. When evaluating a given information system according to the *Common Criteria*, a *Security Target (ST)* is created. The *ST* indicates how an information system is implemented and what (security) functionality it includes. While the *PP*

describes a system in general, the *ST* is a concrete implementation. Hence it is possible to create a *ST* according to a *PP*.

However, as this work is concerned about assuring the correct functionality of information systems, only the third part is used. The assurance requirements and thus the *EAL* levels are utilized to specify the goal of this thesis as discussed in section 1.1. Further it is used in the validation (chapter 6) to evaluate if the approach can fulfill the given requirements. The other parts of the *Common Criteria* are considered in this case as not relevant.

2.1.2 ISO/IEC 27002 - Code of Practice for Information Security Management

ISO/IEC-27002 [ISO05c] evolved from *ISO/IEC-17799* [ISO05a] which further originates from *BS 7799-1* [BS05a]. It includes best practices of information security management. The standard defines a checklist of security requirements grouped into security controls. If an organizations aims to be certified against this standard, it must fulfill the mentioned security requirements. In [Zuc05], a short overview of the security controls is given:

Security Policy mentions that policies provide direction and support for information security

Security Organization describes how the internal security structure should be organized. In addition, security for third party access and outsourcing in general is covered

Asset classification and control deals with the classification of assets and information. This classification should provide the basis for the protection.

Personnel security covers jobs definitions and resources. It also describes user training to build awareness. Finally it deals with reporting obligations for the personnel of security incident response.

Physical environmental security describes what to consider when physically securing the area, the equipment and the workplace.

Communications and operations management provides a checklist for operational security, security planing, software protection, housekeeping, network security, media handling and information exchange. It is mostly about technical security measures.

Access Control describes how to control and monitor the access to information. It also mentions requirements for the management of AC and the user.

Systems development maintenance provides a checklist of important points for developing and maintaining secure information systems.

Business continuity management shows what a continuity plan should contain.

Compliance mentions the importance of investigating and incorporating the legal environment. It also indicates the importance of reviews and audits. [Zuc05]

For this work some of the security controls are taken into account as they match the scope of this thesis. The *Asset classification and control* is a part which influences the functional security testing approach from the very beginning, as we discuss in chapter 4. This is of special importance for this work, as the mentioned security control is already used by *TeliaSonera* and an integration into the approach facilitates an integration into *TeliaSonera*'s processes. Further the *Systems development maintenance* control has an impact on this thesis. As the approach is situated in the software and systems development area, this category is influential during its whole creation process. Regarding the content of this control it applies mainly for the creation of test cases not for the the creation of the approach's structure. However, it is used as guideline, which we consider as relevant for this work.

As it can be seen, some of the security controls are not used for developing the functional security testing approach. This is simply caused due to their inapplicability for the given scope.

2.1.3 ISO/IEC 27001 - Specification for an Information Security Management System

The security management standard *ISO/IEC-27001* [ISO05b] evolved from another security standard named *BS 7799-2* [BS05b]. Further *BS 7799-3* [BS06] is aligned with this ISO/IEC standard. *ISO/IEC-27001* contains information about requirements for managing information security in an organization. Hence it is possible to evaluate organizations in respect to their compliance to *ISO/IEC-27001*. Further this can lead to a certification against the standard. This standard has some cross-references to *ISO/IEC 27002*, which is used by *TeliaSonera*. A certification against *ISO/IEC 27001* is currently out of scope for *TeliaSonera*. Hence, to avoid incompatibility with *TeliaSonera*'s structures, *ISO/IEC 27001* is left beside for this work. However, it is used for term definition. Still, due to the dependencies between *ISO/IEC 27001* and *ISO/IEC 27002* a later integration of *ISO/IEC 27001* into the approach can build on a solid base.

2.1.4 IEEE Std 829-1998 - Software Test Documentation

The *IEEE Std 829-1998* [ISO98] testing standard describes a way of documenting software tests. Further it shows which documents are required for software tests and how they are related to each other. As the organization and management of tests is a major issue in this work, this standard is integrated. It has to be mentioned that *IEEE Std 829* is not meant to be applied without adaption. This is further discussed in [KFN99]. Hence not the whole standard, but relevant parts of it are utilized to create a compliant data structure for the functional security testing approach. We further discuss the use of *IEEE Std 829* for this work in section 4.3.

2.1.5 ISO/IEC 21827 - Systems Security Engineering - Capability Maturity Model

This standard is often referred to as *Systems Security Engineering Capability Maturity Model (SSE-CMM)* [Pro02], [SSE]. This maturity model can be used to evaluate the capability of an organization to use a secure process for developing (information) systems. Further the term *maturity* refers to the quality level of the process from a security perspective. The standard is interrelated to the *Capability Maturity Model (CMM)* [PCCW93], which has evolved to the *Capability Maturity Model Integration* [Tea06]. The SSE-CMM includes the *Process Area (PA) 11 - Verify and Validate Security*. As the name indicates this process area is concerned about the verification and validation of security in the development process. Therefore it includes a list of best practices which should be applied by organizations. The overall goals of these practices are described in [Pro02]:

- Solutions meet security requirements
- Solutions meet the customer's operational security needs

To assure that systems meet their specified security functionality is a major target of the functional security testing approach. To meet the appropriate security level, the approach will use some methods provided by *TeliaSonera* (see section 4.1). Hence it can be said that these goals will influence the development of the functional security testing approach.

2.1.6 ISO/IEC 20000 - IT Infrastructure Library

The *IT Infrastructure Library (ITIL)* is

> a public framework that described Best Practice in IT service management. [CHR+07]

The aim of this standard is to collect best practice for IT service management and further to provide organizations with this knowledge. Thereby the major benefits for organizations are according to [CHR+07]:

- increased user and customer satisfaction with IT services
- improved service availability, directly leading to increased business profits and revenue
- financial savings from reduced rework, lost time, improved resource management and usage
- improved time to market for new products and services
- improved decision making and optimized risk

When considering the above mentioned advantages from using ITIL, it can be seen that this standard has a strong focus on the alignment of IT services to the actual business needs of an organization. As information security plays an important role in this alignment concept, the standard is considered as relevant for this thesis. However, due to the limited scope of this work we subsequently go deeper into the ITIL framework to find more specific best practices which can be used in this thesis.

The latest issue currently available is ITIL version 3 [ITI08]. It contains multiple books, where each book aims on a different part of the (holistic) life-cycle of an IT service. These books are [CHR+07]:

- Service Strategy (SS)
- Service Design (SD)
- Service Transition (ST)
- Service Operation (SO)
- Continual Service Improvement (CSI)

These books are dependent on each other as they form the above mentioned life-cycle for IT service management. Figure 2.1 shows these interrelations. Subsequently we discuss relevant parts for this work according to this life-cycle.

In the service strategy book the *Service Warranty* concept is defined. There it is stated that [CHR+07]:

> Service Warranty: How the service is delivered and its fitness for use in terms of availability, capacity, continuity and security.

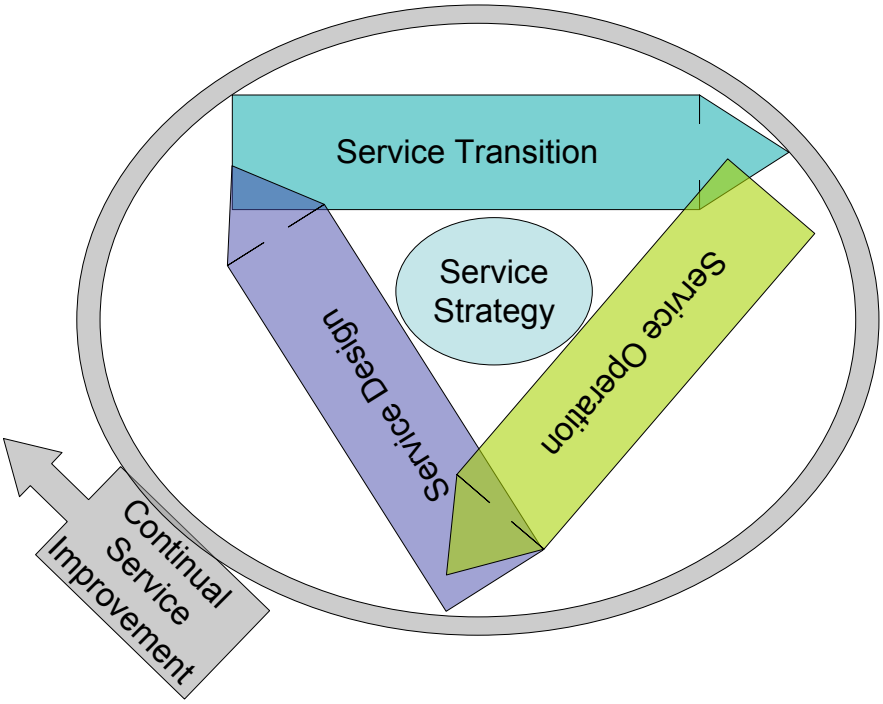

Fig. 2.1 The ITIL service life-cycle, adapted from [CHR+07]

As this work aims on functional security testing of information systems and consequently of the services provided by these systems, their security and availability is a key issue in this thesis. It has to be mentioned that the terms availability and security are used as defined in [Bes]. This has to be taken into account to avoid ambiguity in respect to the diverse usage of these terms in the computer science area. However, the *Service Warranty* concept is influential for this work.

In the service design book two parts of special importance for this thesis are described. These are *Availability Management* and *Information Security Management (ISM)*. The former can further be split up into two key aspects, which are *reactive activities* and *proactive activities*. As this thesis is concerned about functional testing in the pre-operational stages of information systems, only the proactive activities are considered for the development of the functional security testing approach. These activities include

> [...] proactive planning, design, recommendation and improvement of availability. [CHR+07]

However, as already mentioned, ISM is the other key part of the service design book for this work. ISM is about managing information security from a

strategic level to ensure that the level of security is aligned with the business needs of an organization [Bes] [CHR+07]. This further includes activities to assure that [CHR+07]:

- information is available and usable when required (availability)
- information is observed by or disclosed to only those who have a right to know (confidentiality)
- information is complete, accurate and protected against unauthorized modification (integrity)

The proper management of confidentiality, integrity and availability of information is not only requested by this ITIL process, but also of major importance for *TeliaSonera*. Hence the functional security testing approach will pay special attention to the above mentioned activities.

The service transition book is about bridging the gap between designing a service and bringing it to operation. From this book the *Service Validation and Testing* process is influential for this work. This process requires that:

> All services whether in-house or bought-in will need to be tested appropriately, providing validation that business requirements can be met in the full range of expected situations, to the extent of agreed business risk. [CHR+07]

This thesis aims on assuring the correctness of systems, which includes the services they provide, by testing and validating their functionality. Hence the functional security testing approach targets on the same aim as the mentioned ITIL process. Hence this ITIL part influences the approach.

The service operation and continual service improvement books are situated after the service is put into operation. Hence they have no direct influence on the development of the functional security testing approach.

Summarized we observed the ITIL books and their included processes and aims according to their suitability for this work. Thereby we discussed influential and important aspects of these ITIL parts for the further development of the functional security testing approach.

2.2 Verification concepts

Formal verification [Som04] is a method for assuring the correct functionality of a system. This approach relies on an unambiguous representation of the system. Although this method can proof the correctness of its target of evaluation, it needs enormous resources and is nearly impossible to perform due to the complexity of most systems used by *TeliaSonera*. Further this concept is based on access to the algorithm, which is in most cases not available to *TeliaSonera*. Another disadvantage of this verification method

is the fact that side effects (see section 2.5) can not be found. Hence this testing method is not suitable for achieving the target of this thesis.

The most common software and system testing methods use test cases [Som04] which are performed against the target of evaluation. These test cases can have various levels of detail, which reach from low level source code based testing to high level usage tests. In this case the focus lies on high level tests. On the one hand this is caused by the fact that *TeliaSonera* has in most cases no access to the algorithm or source code of its systems. On the other hand such high level tests are performed against already or mostly integrated solutions, which provides the opportunity to detect side effects. As *TeliaSonera* already relies on these techniques for assuring the quality of their diverse and complex systems, it is obvious that we will use them as well. Hence the usage of this method is suitable for the given circumstances and fits to the aim of this work.

Statistical testing [BDG+98] can be used for assuring the quality of software systems. It assumes that faults are distributed according to statistical models and uses these models to describe the distribution of failures and variations of a system in respect to its specification. Hence it relies on mathematics to evaluate the quality of its targets. *TeliaSonera* does currently not use this testing concept. As this approach aims on a sound integration into *TeliaSonera's* practices, the use of this testing method is out of question. However, statistical methods would be only of limited use for functional security tests, which are discussed in chapter 3.

Summarized we discussed some testing methods. Further we chose the most suitable for the integration into *TeliaSonera* and the given requirements. Hence only this testing method is considered in the further parts of this work. It has to be mentioned that the above discussed verification concepts represent only some commonly used and popular ones. Hence there are more verification methods available, but due to the scope of this work and environmental prerequisites determined by *TeliaSonera*, we treated only the listed ones.

2.3 The Software Development Life Cycle

The use of a process, often referred to as Software Development Life Cycle (SDLC), is a widely accepted and deployed approach to create software. This is not only manifest within academic literature, but as well accepted by companies that have implemented such processes in their organizational structures. The activities which are involved in the SDLC can differ dependent on the implementation. However, the underlying intention to establish a

process for the development of software systems, can be considered as similar. *Agile Processes*, as described in [Som04], are not further mentioned due to the scope of this work. Figure 2.2 shows a *waterfall model* of such a SDLC. This process should not be seen as the only existing one, but as a representative example. It contains generic top level activities that can be found in nearly every SDLC as explained in [McG06] or [Som04]. This is regardless of the implementation even if it is a *waterfall*, *spiral* or *prototyping* approach, which are further described in [Som04]. Due to the fact that the aim of this work is on functional security testing, the figure shows the *Scope* and *Depth* of testing according to SDLC activities. These terms are further explained in section 2.5. Following the activities are explained in more detail.

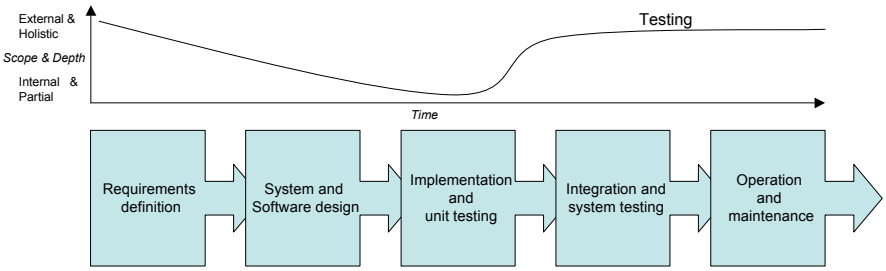

Fig. 2.2 The Software Development Life Cycle (SDLC), adapted from [Som04]

Requirements

At the beginning of the process the requirements for the system are collected and evaluated. During this step not only functional- or security-, but also non-functional requirements, as the latency time of the system or the usability should be specified. Security requirements can be seen as functional or non-functional dependent on the point of view. Due to the orientation of this work the focus lies on functional requirements which are defined in [Som04] as followed:

> These are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. [Som04]

Although it is possible to describe requirements through an unstructured method, it is beneficial for deriving test cases when the given requirements are captured through a (semi-) formal and structured method. This need can be satisfied by the use of *Use Case Diagrams* that are part of the UML Standard, which is defined in [OMG07], or any other exchange format. When using such a method it is indispensable that it is understood by all involved parties within the process, which can get in touch with the requirements. The

more detailed the requirements are elaborated the less ambiguity remains, but the more resources are needed for this step. With a focus on security testing, requirements are valuable as they can be utilized to derive useful test cases with a black box view of the system. This is further explained in section 2.5. These requirements gain value for the derivation of test cases the more detailed they are as later described in section 3.3 through an example.

Design

During the design step in the SDLC the requirements are transfered into a system design, which describes the system from a technical point of view. The design includes the system architecture and specifications about the boundaries or internal system behavior [Som04]. The development of the design can be seen as critical for the process as undetected failures or problems at this step have a significant impact on the information system. As explained in [McG06] undiscovered design failures represent about 50 percent of all security faults and lead to a cost and time consuming redesign in later activities in the SDLC. Due to that test cases, which are derived from the design, are important to discover these failures and reduce costs due to in-security. The created artifacts in the design step are mostly different kinds of diagrams, where UML and its specified design methods can be used for most purposes [OMG07]. Due to the higher level of detail of the specification, the design artifacts provide a solid base for deriving test cases with a stronger internal focus than the test cases derived from the requirements. This focus change can be seen in figure 2.2. Such tests are called *Gray Box Tests* and are further described in section 2.5.

Implementation and unit testing

The implementation step transfers the designed system into executable code according to its specifications. Dependent on the level of detail of the design, different potential risks can be identified. The risks can reach from an implementation bug, like the use of a wrong variable or operation, till a wrong design assumption. Such a wrong design assumption can happen due to incompleteness of the design specification or misunderstood design. Tests can be used to validate and verify the created system components for a given situation. The focus of tests at this level is on the internal structure and behavior of the generated artefacts. This testing approach, with its internal focus, is also called *White Box Testing*, as further described in section 2.5.

Integration and system testing

During the integration step the system is installed into its working environment. When the integration of the system is finished, it can be used for its designated purpose. As all components are coming together the system can be tested as a whole. Due to the integration environmental aspects gain importance for the system and environmental dependent side-effects become

effective. To test a system including its environment *Black Box Tests* are used. As shown in figure 2.2, these tests have a more holistic view on the system than the partial tests during the implementation step.

Operation and maintenance

The operation and maintenance of the system is usually the longest time period within the SDCL. During this step the system is used to satisfy the need that initially triggered the process. This step does not only include the operation itself, but also the improvement of the system [Som04]. This improvement can include the correction of system failures and the adaption to a new environment or changing requirements. Testing at this point has a strong external-, black box view which takes the entire system into account. At this point *Regression Testing* plays a major role, as we further discuss in section 3.4.

2.4 The Secure Development Life Cycle

A problem of the approach in section 2.3 is that security issues often play only a minor role. They are mostly seen as cost drivers with an insurance character, as described in [McG06], rather than as a business enabler, which is explained in [Zuc04]. The insufficiency of security as an explicit and important factor within this process has led to the idea of involving different security approaches and methods during the SDCL. The involvement of explicit security related activities consequently transforms the Software Development Life Cycle into the Secure Development Life Cycle (SDL) [McG06]. As mentioned in [HL02] it is necessary to incorporate these methods into the process, rather than to add security as a feature in the end of the lifecycle. Figure 2.3 shows a graphical representation of the SDL with its touch points for security mechanisms. The showed activities do not exactly correspond with the steps of the SDLC. This shows that the transformation of the SDLC into the SDL can not be seen as static and formal process, as there are many different SDLC process implementations. The advantage of this loosening from a direct and formal transfer method is the generic applicability of the presented methods within the different steps.

Requirements

Typical requirements specify how a system should behave. From a security point of view it can be assumed that someone who wants to break a system does not behave like a normal user, who utilizes it without a malicious intend. This lack in the specification can lead to unmeant and risky system behavior that can be used to compromise the system. According to [Bis03] *Abuse Cases* can be utilized to formally capture this kind of not wanted sys-

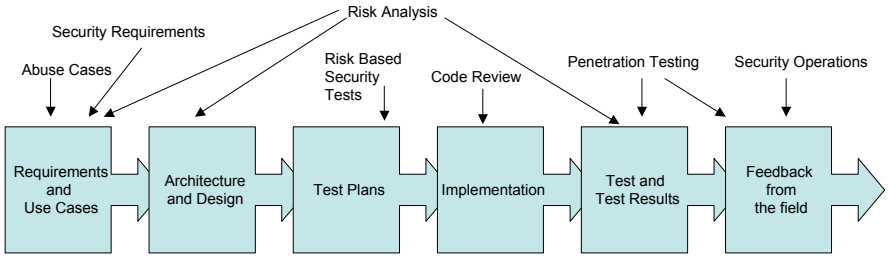

Fig. 2.3 The Secure Development Life Cycle (SDL), adapted from [McG06]

tem behavior. A further description of abuse cases is given in chapter 3.1. The facing of risks through abuse cases can be used for creating awareness for security problems, but they are an insufficient base for testing. It is necessary to take a closer look on each risk and assess it to get the opportunity for an expediently rating. This leads to the need for a *Risk Analysis*, as further described in [McG06]. The risk analysis provides the opportunity to rank identified risks and address them differently. This is necessary due to the limited resources that makes it nearly impossible to pay the same attention to all risks. According to the importance of the identified risks tests can be based on specified requirements which address the risks. For example if the risk of violation of the stored data confidentiality within a system is considered as important, test cases can focus on the requirements which assure the confidentiality of the stored data.

Design

As described in section 2.3 the design phase transforms the requirements into a technical concept. According to [Bis03] design flaws comprehend about 50 percent of all security problems, why security has to be exceedingly kept in mind during this step. To mitigate security related problems during this step it is necessary to keep the ideas of the above described risk analysis in mind and map them to the needs of this activity. This action is described in [McG06] as *Architectural Risk Analysis*. In comparison to the risk analysis made during the requirements engineering, which had a focus on general top level risks, the risk analysis during this step has a focus on more concrete and technical risks. These identified risks can be used as base for what to test for. This enables a risk driven selection of testing areas for a system. These tests are according to the more technical risks soever more technical.

Test Plans

This activity refers in combination with the next step to the *Implementation and Unit Test* part of the SDLC. Due to the identified risks and their mitigation in the precedent steps *Risk Based Security Testing* helps to approve if this mitigation strategy is successful for given cases [McG06]. The

tests should take all defined requirements and anti requirements (further explained in section 3.1), into account. Compared to later life cycle testing, the tests in this step refer to units or modules of a system instead of the entire system. According to [Dij70] a minimization of the probability of failures in parts of a system has a positive impact on the likelihood of failures in the whole system. Hence the security is increased. It has to be mentioned that the tests on this level can not be taken as an approval for the fulfillment of requirements of the integrated and working system. This is caused by the fact that failures may occur due to integration aspects.

Implementation

As previously explained this activity is part of the *Implementation and Unit Test* step in the SDLC. During the implementation many failures can occur due to the call of unsafe methods or the wrong use of language constructs. *Code Review* and *Static Analysis* can be utilized to screen the code to identify and solve such failures [CW07]. In combination with the above mentioned *Architectural Risk Analysis*, the code review takes care about most bugs and flaws during the development of a software system. The code review is not a check for correctness of the code in regard to the language specification, which is done by the compiler or other interpreters. This requirement must at least be fulfilled to meet the premises for the review. The mentioned method analyses the code for misuse of the language within its specification. Static analysis itself can be seen as kind of a testing method where the knowledge database about the known flaws and bugs contains the test cases which are matched against the source code. These tests are focused on internal system details and mostly conducted through semi-automatic tools. An example for such a tool can be found in [FSI].

Tests

In reference to the SDLC it has to be mentioned that this step refers to the *Integration and System Testing* activity. The execution of tests and the observation of the according results at this late part of the development process forms the possibility to assure the quality and functional correctness of the system in its real environment. In comparison to the tests executed during the test plans step, this tests focus on the integration and interaction of the system with its environment. The aim is to ensure that the system behaves as intended. The early specified top level requirements can be used to create the test cases, as it is possible during the test plans step. A testing method which is often utilized during this step is *Penetration Testing* [McG06]. This method is further explained in section 3.2.

Feedback from the field

At this point the software is already shipped to the customer and works in its environment, which refers to the *Operation and Maintenance* activity of the SDLC. Typically security enhancements of the systems are now only

done reactive to discovered vulnerabilities. Mostly patches are used to solve these problems. Due to the fact that built in software security does not play a major role in most current systems, many security tasks are delegated to the environment and other systems. These systems can be for example firewalls or intrusion detection systems. This is just reactive as it is an attempt to make a system secure due to the control of interactions with its environment, even though it would be better if the system itself is designed and implemented as internally secure from the bottom up [McG06]. Due to these facts security issues must be mentioned from the very first beginning of the development process until its end.

2.5 Testing

Testing aims to verify the behavior of a system for certain, predefined test cases. Due to the strong dependence on given cases, testing can never be used to demonstrate the total correctness of a system. Edsger W. Dijkstra says in [Dij70]:

> Program testing can be used to show the presence of bugs, but never to show their absence! [Dij70]

This quote clearly defines the limitation of security for assurance of testing.

As shown in figure 2.4, the implementation of a system and its requirements can be viewed as two different parts, which must not be congruent. This can be caused by an improper fulfillment of the requirements, which are not implemented exactly as specified or intended by the creator, or if the implementation includes effects which were not considered in advance. The problems caused by this divergency between the requirements and the implemented system can be categorized as followed:

- Missing or incorrect fulfillment of the requirements
- Side effects, which are not considered through the requirements

Due to the requirements it is possible to test the specification as described above, and hence identify where the requirements are not fulfilled. A more critical problem is latent through side effects, which can not be found through requirements based testing methods, as they are beyond the specification. Due to the unknown and untested behavior of the observed system in those areas, the side effects can be used by attackers to compromise the system.

To discover both types of divergency, we present two different testing approaches. One of the testing methods is functional testing. As described in

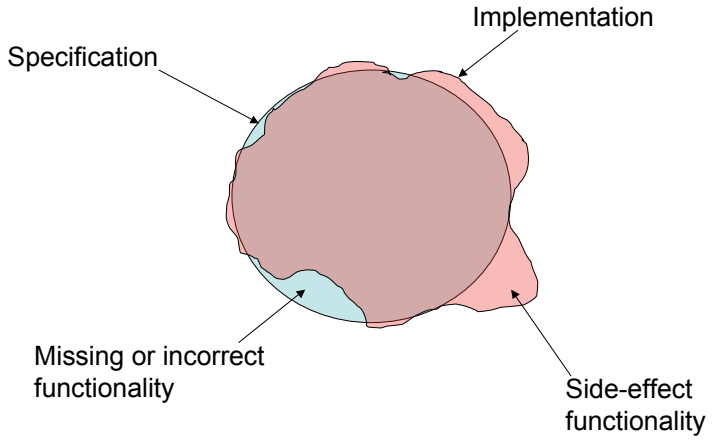

Fig. 2.4 The aim of Testing, adapted from [Tho03]

[McG06] or [Tho03], functional test cases can be created for specified requirements and hence can indicate that those requirements are fulfilled as specified. Functional testing can be used to examine the system and focus on the fulfillment of the requirements. Hence it can expose missing or incorrect functionality within the specification, as further described in section 3.3. The other testing approach is penetration testing, which aims to find unspecified side effects that can be used to provoke unwanted system behavior. This is further discussed in section 3.2.

For the further work it is beneficial to categorize and classify testing approaches. The aim of testing, as shown in figure 2.4, is alone not sufficient to classify testing methods as it pays only attention to side effects or requirements. To differentiate testing methods, the dimensions *Scope* and *Depth* should be added. In this case the aim of the testing approach is a parameter within each dimension that further specifies the testing approach according to its focus. The dimensions have been choosen from the Common Criteria [ISO07], where the *ATE_DPT* family specifies criteria for testing. We further describe the dimensions in the subsections.

Figure 2.5 shows how testing approaches can be arranged within this two dimensional space. This figure is influenced by [Som04], [Jan05], [McG06] and [ISO07]. The influence of the aim of the testing method, which can be either related to the requirements or to the penetration of the system, is not stated out explicitly as it can not be seen as separate dimension. This is later described in section 3.3 and 3.2. The three shown types of testing approaches are *Component Testing*, *Integration Testing* and *Sys-*

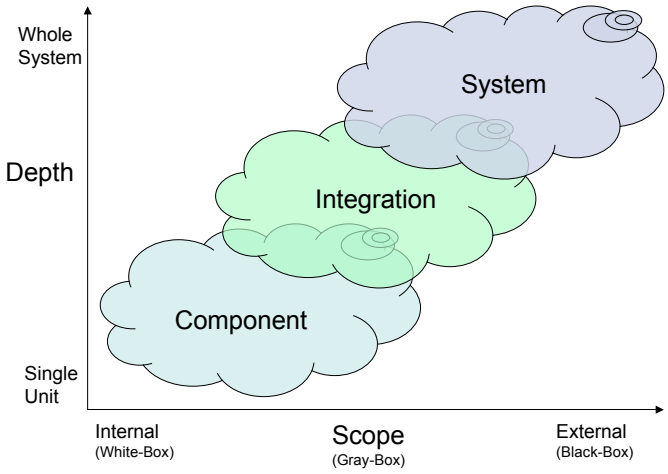

Fig. 2.5 Categorization of testing approaches

tem Testing. In the following subsections we discuss every dimension and its meaning in detail.

2.5.1 Scope of testing

To compare testing approaches it is necessary to take their scope in respect to the system into account. The scope has two extremes, which are the inside-out view, where the internal construction is the starting point and an outside-in view, where the system is seen encapsulated and only the input and output can be evaluated. These two views of testing are also called *White Box Testing*, which refers to the internal inside-out methodology and *Black Box Testing*, which corresponds to the outside-in, or external view.

White Box Testing plays a major role during the SDL within the *Test Plans* and *Implementation* step. Due to that this testing method assumes that access to the internal system is given. This targets especially on access to the source code. During a white box test the system is analyzed internally. This enables the tester to use knowledge about how the system works and operates as described in [Jan05]:

> White box testing includes analyzing data flow, control flow, information flow, coding practices, and exception and error handling within the system, to test the intended and unintended software behavior. [Jan05]

The tests can not only cover cases which retrace if the implementation follows the predetermined design and architecture, but also aspects like the proper implementation of security functionality or the quality of the code. The level of this kind of testing is mostly modular or unit based, where parts of a system are tested in order to approve their correctness. These tests demand a consolidated knowledge of the tester about the system, the implementation environment and of course of security issues. As the scope is just a categorization for tests it is possible to create and perform a test which takes the whole system from an internal, white box point of view, into account. Even though this is possible it is very challenging to conduct such detailed tests for an entire and complex system. This causes the fact that tests often have to find a balance between their scope and depth in accordance to the limited availability of resources.

Black Box Testing follows the outside-in approach and hence does not take the internal structure of the system into account to create the test cases and conduct the tests. A definition of black box testing is given in [KFN99]:

> [...] black box testing, in which the program is treated as a black box. You can't see into it. The tester (or programmer) feeds input data, observes output data, but does not know, or pretends not to know how the program works. [KFN99]

As foundation for the analyzation of the system an exclusive outside oriented view is considered for the test cases, which makes black box testing useful for systems which cannot be internally analyzed due to technical or legal limitations. Compared to the described white box tests these tests are primarily performed at the end of the development life cycle against the entire and integrated system as described in [McG06].

The distinction of black- and white box testing is only possible in theory, as both contain elements of the other. The differentiation can be done in respect to the focus of the test. This is caused by the fact that there are different levels of detail of the internal view of a system. Not even the source code is the representation of the real executed system because a source code analyzation can be seen as a black box analyzation of the real executed machine code. It is not simple to draw a line where the internal white box view ends and the external black box view begins. This differentiation problem leads to the introduction of a new term, named *Gray Box Testing*. This is further described in [Jan05]. The gray box testing approache lies between the internal white box- and the external black box testing. Gray box testing is an important part within the security testing area. Some security testing approaches start with a black box view of the system and then drill deeper into specific system details, as used in the white box analysis. Due to the recently described facts pure black box or white box testing approaches hardly exist in reality as at least some abstraction level or system detail is used for

conducting useful tests. To make the black box- and white box classification usable as terms during this work they cannot be seen as strict classification, but as fuzzy ones as the gray box area itself.

2.5.2 Depth of testing

The depth dimension refers to the different levels of detail a system can be observed on. This leads to a different usage of tests during the development cycle according to their depth. Unit or also called component tests, which are performed against a part of a system, are mostly utilized earlier in the development process during the creation of artifacts. System tests aim at the testing of the entire, holistic system in its environment and hence can be as recently used as the system is completed.

Component testing is strongly related to white box testing and often seen as correlating testing approach, as mentioned in [Som04]. System testing on the other hand is related to black box testing, as the system is seen as a monolithic and homogenous one. As the separation of testing approaches within the scope dimension the separation on this axis can also not be seen as straight. Due to this, the term *Module testing* is used, which aims on the fuzzy border between the extremes of the dimension.

2.5.3 The software testing process

Testing can not be seen as a single action, as it is a process that contains multiple activities and artifacts. Figure 2.6 shows a top level view of a process which states out how testing can be structured. The initial step in the process

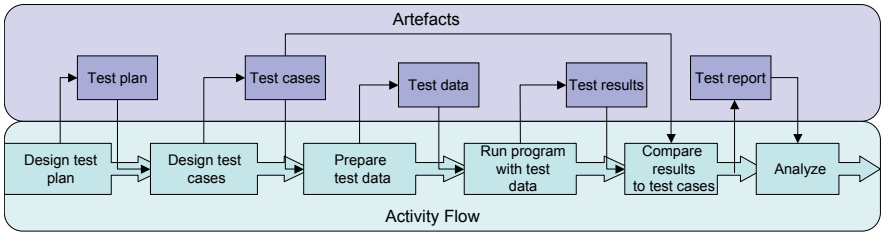

Fig. 2.6 The Software Testing process, adapted from [Som04]

is the creation of a test plan, where the further testing activities are organized and determined. During the second step test cases are created. These cases cover the system that has to be tested. As not explicitly mentioned in figure 2.6, the development and design of test cases should be driven by a risk analysis and a general testing strategy [Jan05]. Due to the kind of the created test, different methods to perform the analyzation are taken into account. These methods can be determined due to the different placement within the above described dimensions and due to the aim which is described by the additional parameter as mentioned earlier in this chapter. The created cases should further include facts like preconditions, inputs, results and the process of how to perform them, as described in [Jan05]. They should provide a source for the *Test data*, which have to be prepared in the next step. When the test data are created the test cases can be conducted. This leads to the *Test results*. These results are further interpreted and compared in a report according to their expected results, as earlier defined in the process by the test cases. According to that the *Test report* is created, which is analyzed according to its impact on the development process. The testing process presented in figure 2.6 has a lack of completeness, as the influence of the analyzation on the development cycle is not shown explicitly. However, it has to be mentioned that the analyzation, especially if it states out unwanted behavior of the system, has to be taken into account for the development cycle.

Chapter 3
System Security Testing

This chapter gives an overview of security testing methods for software based systems and describes the reasons for differentiating between security- and traditional system testing. The first section describes which differentiation criteria can be taken into account to distinguish between security- and other types of failures. As it is shown in section 3.1 these different types of failures have a different impact on the system. Hence different testing methods are needed and a differentiation between security and traditional testing methods is necessary. Following a substantiation for the importance of security testing is given. Due to that security testing approaches must be different than traditional testing approaches. After this delineation from traditional testing, different kinds of security testing approaches and their differentiation criteria are presented. At first penetration testing is described. Based on that functional security testing, one of the core parts of this work, is explained in detail. A definition of functional security testing and its used methods is presented. This provides differentiation criteria between functional- and other- security testing methods. Finaly regression testing and its relation to functional testing is described.

3.1 Security- and Traditional Testing

Security testing is different from traditional testing, as the former has to cope with an attacker who actively searches for vulnerabilities [WT03], [HL02], [CJ06], [McG06], [MR05] to break a system. Normal testing instead assumes a non-malicious usage of the system for its designated purpose. Therefore security testing must not only assure the proper functionality of the requirements, but also has to pay attention that not more than the spec-

ified requirement is implemented to avoid side effects as described in section 2.5. Relating to [MR05] we find:

- Security testing, checking for negative requirements, emphasizes what an application should not do rather than what it should do, which requires side behavior testing
- Security requirements are ambiguous and the alternatives are hard to enumerate, which makes it hard to find appropriate test cases

As shown by these statements security requirements are different from traditional requirements and hence need to be treated in a special way. Still, there is a need for assuring the functional correctness of security requirements, which is further discussed in [Mea06]. Both points refer to the problems of security testing. As many security requirements have no explicit related test case it is very challenging to assure that they are fulfilled. This problem of how to test security requirements states out a difference against most traditional requirements, which have a certain target that can be tested for. To handle the problem of untestable security requirements *Abuse Cases* can be used. As mentioned in section 2.4 within the SDL, abuse cases describe how an attacker could try to jeopardize the system. To create this kind of use cases the originator must have an iniquitous point of view to cover the aspects of this method. Abuse cases can be used to specify system behavior as described by the following quote:

> In some cases, the functional requirements may also explicitly state what the system should not do. [Som04]

These kind of specifications are often called *Anti Requirements* or *Negative Requirements* ([Bis03], [McG06] and [MR05]). They specify how a system should not behave and hence create a more precise delineation of the system. A problem with negative requirements is stated out in [MR05]:

> The mapping of requirements to specific software artifacts is problematic for a requirement such as ”no module may be susceptible to buffer overflows” [...] [MR05].

This shows the main problem of negative requirements, which is the fact that these requirements may not occur in any part of the system. Due to that they cannot be tested properly. Another disadvantage of this *Black Listing* approach is that it is always hard to define every forbidden and unwanted state or action a system should not accept. This generic problem of black listing further decreases the benefit from the use of negative requirements as a basis for testing.

However, they can still have a positive impact on the security of a system. As further described in [MR05] these negative requirements can be taken to explicitly show risks and problems. This leads to the opportunity of directly addressing them through positive requirements. The quoted negative

requirement can lead to a positive requirement that specifies that all input data must be checked properly with attention to their length and content, as wrong and unchecked input data are one major cause for buffer overflows [McG06]. Although if some information gets lost during the transformation from a negative to a positive requirement, it can be used to create awareness for the risk and hence mitigate it as good as possible. Another advantage of anti requirements is that they can be specified early in the development process. This makes it easy and cost saving to take them into account for deriving test cases.

Security requirements, which are specified to protect the system from threats, build a good source for the creation of negative requirements as their malfunction and abuse by attackers can be directly transformed into them. Due to the explicit description of the problem it is possible to include it in the testing process and hence establish a higher level of security. The combination of both types of requirements makes it possible to set up a more precise top level specification for systems that can be used as a base for the further development process.

Summarized it is clear that negative requirements can help to create a more precise and secure description of the requirements for the whole system, but only the awareness of a risk does not consequently lead to its successful mitigation. Hence security tests are needed.

3.2 Penetration Testing

Penetration Testing is a testing method, which targets at the breakup of a system. To achieve that, the system is regarded from an attackers point of view with a malicious intend. An overview of methods, which are used for this kind of testing, is given in [HM04]. This testing approach cannot be used to assure that the security functionality of a system behaves as specified, but to conduct tests with an aim as explained in [vW07]:

> [...] ”penetration testing” refers to testing the security of a computer system and/or software application by attempting to compromise its security, and in particular the security of the underlying operating system and network component configurations. [vW07]

The detection of faults and defects, as further mentioned in [Som04], is the primary concern of this testing approach. According to figure 2.4 penetration testing focus on the side effects of an implementation, rather than the requirements. Figure 3.1 shows how penetration testing is situated in relation to the implementation and requirements of a system. In comparison to later described functional testing it only takes parts of the system into account. Within these parts it aims on deeper inquiries for possible side effects.

Due to that this testing approach is able to discover more side effects than

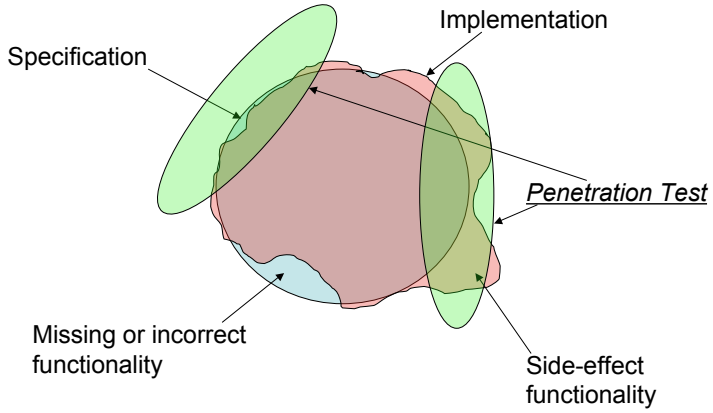

Fig. 3.1 The aim of penetration Testing, adapted from [Tho03]

the later described functional security testing. Hence this testing approach compromises the system by the exploitation of unknown and unwanted behavior. To discover such side effects the requirements can not be utilized as input, but a malicious intend must be used to derive the tests for this testing approach. As mentioned in [vW07] penetration tests are often conducted through the support of tools that help to compromise a system. This tools have only an external view of the system. When a potential vulnerability is detected the part of the system which is affected is examined in more detail. This is usually done by reverse engineering and de-compilation which reproduces the source code or any other more detailed representation of the executed part [HM04]. Hence this provides a closer examination and makes the system behavior comprehensible. Due to that new opportunities to compromise the system are gradually widened.

This leads to the problem that a penetration test only points out the discovered faults and can not directly address the source of problems within a system. This would require a more internal and holistic orientation as explained in [vW07]. It must also be mentioned that penetration tests are, according to [McG06], mainly used as late life cycle tests. Hence they are not as cost efficient as earlier life cycle tests [MR05].

3.3 Functional Security Testing

Functional Security Testing, which is addressed during this section, can be used to assure that security functionality behaves as specified by the requirements. A definition of security testing is given in [PM04]:

> Thus security testing must necessarily involve two diverse approaches:
>
> 1. testing security mechanisms to ensure that their functionality is properly implemented, and
> 2. performing risk-based security testing motivated by understanding and simulating the attacker's approach

This definition for security testing can be seen as too extensive and fuzzy when it comes down to *functional* security testing. Due to that a more detailed and focused definition is given in a following subsection. Meanwhile the shown definition can be used.

Functional security tests can approve the correct implementation and fulfillment of specified requirements, as shown in figure 3.2. According to the

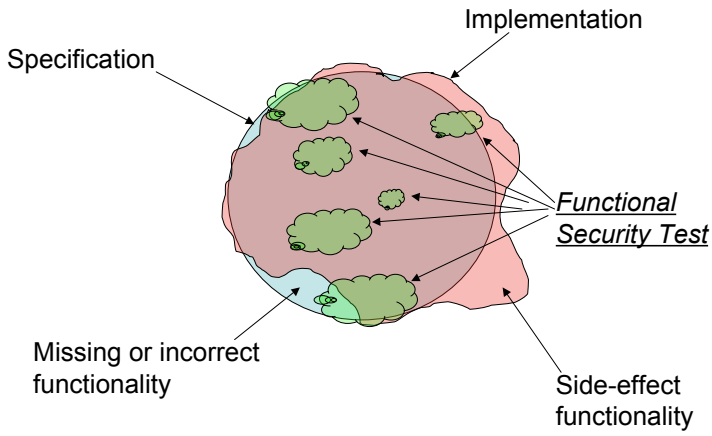

Fig. 3.2 The aim of functional Testing, adapted from [Tho03]

previously in this work described aim of a testing approach in figure 2.4, figure 3.2 extends the aim through the influence of functional testing. It can be seen in the graphical representation of functional security testing, that this testing method targets on the requirements of a system. In comparison to penetration testing, the aim of functional testing is on specified requirements and slightly beyond them. This slight extension of the focus from pure requirements is caused by two reasons. On the one hand it is beneficial

due to the fact that this provides to catch the low hanging fruits of discovering simple side effects. On the other hand it is necessary trough the mostly insufficient description of security requirements by their specification. An exclusive orientation on the given functional requirements specification for deriving functional test cases is questionable and not useful in most cases. This is caused by the fact that requirements are often not as exact described as they must be for directly deriving test cases from them. Due to that they need to be put into context in which they need to be interpreted. For example the requirement for an user authentication is often specified as followed:

- Users should be authenticated through a login

This requirement would not be very useful if a test is strictly derived from the given information, as nothing is stated out neither about wrong or missing inputs, nor about the behavior of the system in case of failure. A correct specification of such a requirement is given in [Tel02]:

- Users should be authenticated through a login
- Login (correct username, correct password) → success
- Login (wrong username, wrong password) → failure
- Login (wrong username, correct password) → failure
- Login (correct username, wrong password) → failure
- Error messages for unsuccessful logins should be created
- Respond time for successful versus all types of unsuccessful logins should be checked
- Transfered data during login process must be encrypted
- Automatic logout after a certain (idle) time
- No default users and passwords should exist

This complex expansion of the requirement has to be made to specify only typical and common expected behavior from an authentication mechanism. If some extra behavior, as for example the check of a password change mechanism or a strength verification for passwords is wanted, the list has to be extended. It can be seen that an expansion of requirements as even described is uncommon and unrealistic to expect during the requirements engineering. Functional test cases have to pay attention to this fact. Due to that it is necessary that during the creation of the test cases, the actual requirements must be interpreted within their context. Hence the extended requirements are taken to derive the test cases and their testing procedures. Negative requirements, as earlier described during this chapter, play an important role for functional tests and can be transformed into positive ones. This has already been demonstrated during this work in section 3.1. According to this, the positive requirements can be extended and described as mentioned above. Another important point, which has to be mentioned, is the use of functional

testing within the development process. Compared to penetration testing and according to [McG06], functional testing has often a more internal view and is due to that conducted earlier within the life cycle. To conduct functional security tests several techniques exist, where the most useful are described in the next subsection.

3.3.1 Functional Security Testing Techniques

In [TT07] some testing techniques, which are relevant for functional security testing, are presented. In contrast to the list of techniques in [MR05] and [Per91], this list has already been checked for their relevance and suitability for functional security tests [TT07]:

Requirements-based testing
This technique takes the requirements of a system into account and derives test cases through them. It enables the coverage of specified functionality and is useful as technique to assure that the requirements of a system are fulfilled.

Equivalence partitioning
Systems have often inputs which are treated the same, as for example the purchase of a CD or a DVD within a shopping system. When utilizing the equivalence partitioning technique multiple test cases are derived for each of these partitions. This includes valid or invalid input as further mentioned in [TT07].

Boundary value analysis
This testing technique refers to the test of a systems behavior when it is pushed to its boundaries. Suitable test cases for this technique examine the behavior of the system at its boundaries and slightly around them. This is caused due to the fact that boundary cases often discover unintended behavior as described in [CJ06].

Robustness and fault tolerance testing
This technique aims to test the behavior of a system far beyond its boundaries. A standard example for this is the use of long or oversized inputs to cause a buffer overflow as explained in [MR05].

Usage based and use-case based testing
As use cases specify how a system will be used during its operation it is obvious that these specifications can be used as a basis to derive test cases. For security testing especially abuse cases and negative requirements, which were both earlier described in this chapter, play an important role when it comes to deriving test cases through them.

3.3.2 Definition of Functional Security Testing

As already mentioned, the above given definition of security testing can be seen as too extensive and not specific enough for the aim of this work. As we did not find any definition for functional security testing, one is given now. It has to be mentioned that this definition is strongly influenced by *Albin Zuccato*, a supervisor of this thesis. Functional security testing can be defined as followed:

Functional security tests aim on testing the

- correctness of relevant spots
- and around the boarders of a requirement
- without malicious intend

In comparison to penetration testing (section 3.2), functional security testing has its aim on requirements and only slightly beyond them. This differs from penetration testing, which has its focus on side effects and only parts of the system, rather than the fulfillment of requirements. Another differentiation criteria is the intend of the testing approaches. While penetration testing has a malicious intend and aims on the breakup of a system through compromising parts of it, functional security testing aims on the correct fulfillment of the system's requirements specification. This is also expressed through the use of the above described functional security test techniques. Their use is implied due to the non-malicious intend of functional security testing.

It has to be stated out that there may be of course cases where the presented definition and differentiation criteria have a lack of plain correctness, but it has to be kept in mind that this is caused due to the fuzzy borders when working on and beyond the edge of common accepted knowledge.

3.4 Regression Testing in Security

Regression Tests aim at the reuse of already conducted tests to assure that the use of updates or the extension of given systems does not influence the original and already tested functionality of the system. In [KFN99] this kind of testing approach is defined as followed:

> The term *regression testing* is [...] the idea of reusing old tests [...] [KFN99]

Regression testing can be seen from two different points of view, which can be distinguished by the level of completeness in regard to the specified

requirements of a system. According to [KFN99] the different approaches and their included procedures can be defined as followed:

- Finding an error → fixing it → repeating the test that exposed the problem in the first place
- Finding an error → fixing it → executing a standard series of tests to make sure that the change did not disturb anything else

The first definition covers only tests for a specific part of a system, in which a failure has been discovered or a change has been made, while the second definition has a more holistic approach of testing. The need for regression tests is explained in [JC83], as fewer than half of the fixes solve the problem after a first test. Following the focus of this work, the second definition is used.

Regression tests provide a possibility to reuse security test cases for systems which are improved and refined and therefore form new systems. This requires the opportunity to define a classification of tests according to their importance and coverage of requirements. Those tests can then be reused in a regression test to assure that the refined system does not disturb the initial functionality of the base system.

According to the aim of a testing approach as described in figure 2.4, the regression test can be regarded as shown in figure 3.3. As it can be seen, re-

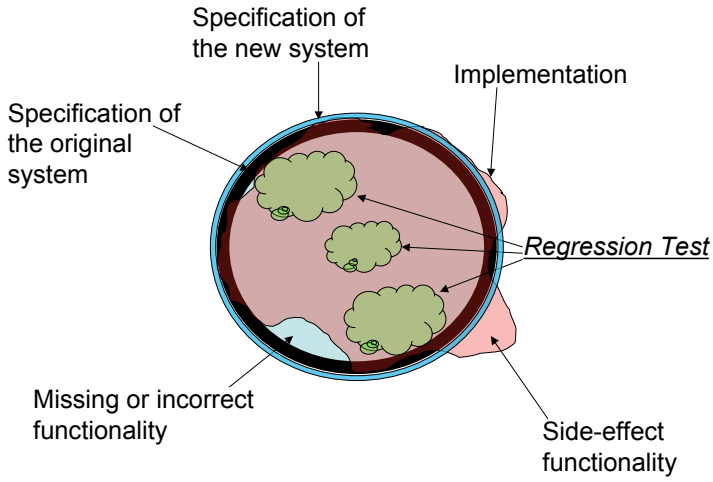

Fig. 3.3 The aim of regression Testing, adapted from [Tho03]

gression tests do not cover the complete system, as done by functional tests. The aim lies on few critical and important parts of the entire specification,

which are taken as representative examples for the base system. This leads to a reduction of inevitable test cases for the new system. An important decision for the creation of regression tests is the selection of representative tests for the functionality of a system. Although this is critical and according to [KFN99] also challenging, it has do be done for most systems due to the large number of available tests which are necessary for an entire test of the system. This problem could be partially solved by the use of automated tests [DRP99], as they significantly reduce the execution time. Since this is not the topic of this work it is not further considered here, but a detailed discussion can be found in [KFN99].

Chapter 4
The Functional Security Testing Approach

To test a systems' compliance with its requirements it is necessary to link them to their test cases. Indeed this seems obvious, it is unfortunately not trivial, as we discuss in this chapter. Requirements are frequently tested by different test cases. Further a test case can be reused for testing a range of requirements. Hence an approach for an unambiguous mapping between requirements and test cases is needed. Such an approach should support the testing process. It should not only pay attention to the static structure, but also to the necessary testing activities. These two are indispensable to capture and manage the complex relationship between requirements and test cases. The structure and the activities we propose are provided in this chapter by a database and a process, which are derived according to the requirements for such an approach.

At first this chapter gives an overview of how security requirements can be derived due to protection needs. As this is a common practice in *TeliaSonera* it can be seen as mandatory condition for the development of the approach. Further this refers to the asset classification security control in [ISO05c]. Following the process of the approach in respect to an existing testing process is shown. Subsequently we introduce the necessary structure for a test case according to [ISO98]. According to the static structure a database is presented which is used to manage and organize test cases as well as their related requirements. During the description of this database, incremental steps lead to a continuous refinement of the database structure. First, the requirements for the database are collected and top level entities are identified. According to that a top level schema is developed, which pays attention to the relationship between the identified entities and the holistic structure for the test cases. This leads to a further refinement through the derivation of a relational schema. The schema has its focus on the implementation of the structure of a test case and its relations within a database. The last step during

the creation of the database is presented by the implementation subsection. Within this part the relational schema is transformed into executable SQL code. The target system for this code is MySQL [MA07]. Hence attention is payed to system specific features provided by the used system. These features are used to enhance the quality of the created database schema. It has to be mentioned that the presented database structure is developed with an eye on flexibility. This is done in order to enable a long term future usage and to provide support for rare and uncommon cases. However, it also enables the management of a wide range of standard test cases and requirements.

Finally, we introduce a process for the creation of content for the functional security testing approach. As the enrichment of the approach by quantitative test data is a critical success factor, we see this as a necessity to provide long-term value for *TeliaSonera*.

4.1 TeliaSonera's Security Mechanism Reference Table

To structure security mechanisms and requirements within *TeliaSonera* the *Security Mechanism Reference Table (SMRT)* is used. This table provides a classification of mechanisms according to their qualitative and quantitative security properties. The qualitative part refers to the kind of given protection, while the quantitative aspect refers to the strength of provided protection by the mechanism. The qualitative component of a mechanism is further split up due to the C, I, and A criteria, which is further explained in [Gol06] and [PP03]. The quantitative factor is organized in several protection levels, which state the strength of protection. These security- or protection- levels are further described in section 5.1. Figure 4.1 shows a simplified version of the SMRT. It has to be mentioned that the complete table cannot be published due to security concerns of *TeliaSonera*, but a hunch of how it is used is given now.

While developing a new information system the different information types, which are processed by it, are classified. The classification indicates the protection need of information derived in the business domain. To translate the protection need into the technical domains security requirements, one has to use the C, I and A protection needs and find the relevant security mechanism in the SMRT. The strength given in the business need, is deterministic for the mechanism strength. The SMRT is closely related to the *Asset classification and control* security control from *ISO/IEC 27002* [ISO05c], the *Verify and Validate Security* process area from [Pro02] and some *ITIL* best practices [CHR$^+$07]. We already discussed these issues in

chapter 1. Summarized it can be said that the SMRT closes the gap between identifying protection needs and finding mechanisms to satisfy them.

Security mechanism		Confidentiality	Integrity	Availability
Environment dependent	Authentication	X	X	
	Access Control	X	X	
	Data integrity transfer protection		X	
	Data confidentiality transfer protection	X		
	Stored data integrity		X	
	...			
Environment independent	Traceability		X	
	Restoreability			X
	Key Management	X	X	X
	...			

Table 4.1 Security Mechanism Reference Table (SMRT) [Tel07b]

As it can be seen in table 4.1, the security mechanisms are divided into two groups, which are named *Environment dependent* and *Environment independent*. The first group contains mechanisms which have to be seen in context of the environment a system is situated in, and the environmental *baseline protection* provided due to this. The second includes mechanisms that are not affected by environmental influences. Due to the environmental impact on the mechanisms within the first group, an identified protection need can be superseded by the environment the system is placed in. This is the baseline protection required by the environment and leads to an homogenous security standard in it. The *Environment independent* security mechanisms group refers to mechanisms, which have to be implemented by systems independent from the baseline protection of its environment.

The concrete usage of the SMRT is now shown in a short example. A new information system is developed and according to its processed information types, a protection at level 2-2-2 (C, I, A) is needed. As the system is supposed to be in a network which provides environment dependent security mechanisms at a level of 2-2-2, the environmental dependent security mechanisms of the system must fulfill at least this level to assure that the *baseline protection* is not violated. In this case the needed level for the *baseline pro-*

tection is equal to the needed level for the system itself. If there would exist a gap, stronger security mechanism have to be used to fulfill the higher protection needs. To achieve an entire protection at level 2-2-2 for the system, the environment independent mechanisms have to be taken into account as well. As these mechanisms are independent from the environmental protection, other mechanisms have to be chosen to fulfill the 2-2-2 requirement. As shown in this example, the SMRT provides a very effective but simple mapping between identified protection needs and security mechanisms to satisfy them.

In order to create an approach for functional security testing that can easily be integrated into the organizational structures and processes within *TeliaSonera*, the SMRT and its underlying ideas are used during the development. The SMRT predefines a way of how to work. Due to that it influences the static- and dynamic- structure of the developed approach. Especially predefined static data like the security levels or the categorization of security mechanisms are heavily influenced by the SMRT (see section 5.1). It provides an easy integration into the processes and structures of *TeliaSonera*, and enables the approach to built-up on practical approved and established concepts.

4.2 The process for Functional Security Testing

When developing an approach for functional security testing it is necessary to state out which activities of the testing process are affected and how the current testing process is supported. Figure 4.1 shows the impact the testing approach has on the standard testing process explained in section 2.5. The

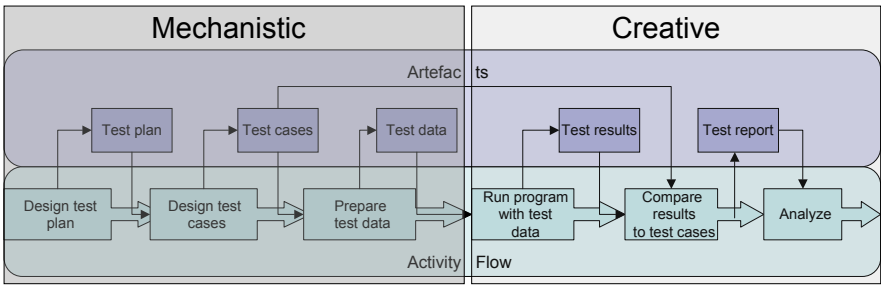

Fig. 4.1 Functional Security Testing Approach Process

process is covered by two different areas, which are *Mechanistic* and *Cre-*

ative. Both refer to the kind of support for the process given by the testing approach. The *Mechanistic* term is used according to an automatization level which is mostly independent from the creative capabilities of the tester who uses the approach in this part of the process. Hence it refers to a level of automatization that accurately prescribes the actions which have to be performed on a detailed level. The term *Creative* is used to show that this part is paid attention to in the testing approach, but its conduction still needs the creative capabilities of the tester. These capabilities are further basic computer- and security knowledge. It has to be mentioned that even in this areas the approach supports the tester and determines many activities. Still, an interpretation of these instructions is needed. The idea for the separation of coverage by the mentioned terms is taken from [Zuc05].

The aim of the testing approach is to have as much as possible automatization of the testing process. According to that the *Mechanistic* coverage is the preferred one. However, a full mechanistic coverage is out of scope of this work as it would not only include the execution of tests, but also the comparison of outcomes to its expected results, as well as an analyzation of these generated data. Hence the gap to reach a full mechanistic coverage can be declared as quite large when it comes to test complex and heterogeneous systems, as it is done by *TeliaSonera*. Subsequently, we discuss the activities and their related artifacts in detail.

Design test plan This activity is supported due to the semi-automatic derivation of test cases for given requirements. Hence it enables the tester to generate the *Test plan* artifact. It has to be mentioned that the test plan does not only consist of the data generated by the approach, as it would just cover some security aspects of the testing process. In addition the *Test Coverage Assessment* [Tel07d] and *Security Test Plan* [Tel07c] documents are used by *TeliaSonera* to embed the given information by the testing approach into the holistic testing plan. This testing plan can include the need for additional tests, as for example to penetrate a system. Further it can also require review activities.

Design test cases According to the created test plan, this activity refers to the crafting of the test cases in order to fulfill it. The approach will provide a number of predefined test cases for given requirements, which are based on the SMRT. Due to that the design of test cases for already tested requirements is not necessary during the process. Hence this activity can be declared as semi-automatical accomplishable for requirements which are already included in the approach.

Prepare test data The preparation of test data must be performed according to the test cases they should be used for. Due to that this activity and thus the resulting *Test data* artifact are both covered by the predefined

test cases, which are created in the previous activity. The merge of the creation of test cases and the preparation of test data by the approach is caused by the need for a tight relation between them to create useful tests. A separation of these activities can be declared as non-beneficial as the merge enables a more comprehensive view of the created test cases and avoids weak test data, which do not actually fit to the test cases. The utilization of weak test data can happen if the tester has a different view on the test case and therefore would create test data which do not reflect the creator's idea. To mitigate this the creation of test data is merged with the creation of the test cases. It has to be mentioned that the actual test data must not be written into a test case, as it is in many cases more useful to make a description of how the test data have to look like. This is caused due to flexibility needs, as it helps to make the tests more adaptable to a specific situation and environment. However, the description of the data must be exact and unambiguous. Hence this activity can still be seen as mechanistic if no new test cases have to be created.

As already mentioned the activities and artifacts are mechanistically covered by the approach till to this point. This coverage can not be classified as full-, but semi- automatic. To close the gap to reach a full automatic processing of these steps it is necessary to formally specify the requirements, which feed as input for the design of the test plan. As this can be seen on the one hand as technical problem, it must be kept in mind that such a need, to formally specify requirements, also includes many stakeholders and can be seen as challenging when it comes down to heterogeneous systems as used by *TeliaSonera*. Further this leads to organizational change on the other hand. This is out of scope of this work as one aim of the approach is to be directly integrable into given structures and processes. However, the remaining activities and artifacts of the process are covered in a more generic way, which puts higher demands on the tester than the steps before. Still, the approach supports every activity and artifact as we discuss now.

Run program with test data This activity refers to the execution of the tests. This part of the process is not fully covered by the approach as it does not include any techniques or information for automatically performing the given test cases. Of course the test cases include information about the testing procedure and a description of how to execute the tests, but none of this information is supposed to be automatically executed by a machine. As it was already described in chapter 1, this would be beyond the scope of this work. The main cause for that is the high diversity of requirements and test cases according to the heterogeneous systems used by *TeliaSonera* and the dependency of tests on the environment. Due do

that some adjustments to these mentioned factors have do be done when performing this activity.

Compare results to test cases Due to the none-automatic execution of the test cases and the not formally specified *Test results*, this activity can not be fully covered. The test cases include evaluation criteria which describe the expected results, but the comparison of the outcomes to these criteria is still left to the tester. Hence the approach provides support for the comparison conducted by humans, and consequently support for the *Test report* artifact.

Analyze The created test report includes information about the tests and their outcomes. The *Analyze* activity is based on this report to derive important information which further influences the top-level development process. The approach covers this activity by providing statistical information about the conducted tests. This statistic is about the successful execution of tests, problems during the conduction and other directly and indirectly relevant factors, as for example the time of execution or duration.

Summarized it can be said that the approach will support every part of the testing process, while some steps are covered in more detail than others. However, it has to be mentioned that even for the *Mechanistic* parts a tester is needed, although they are mostly automatized and do not require a lot of human knowledge or influence.

4.3 The structure of a Functional Security Testcase

To craft functional security test cases the standard [ISO98] is used as reference. Due to the nature of this standard, only relevant parts are taken into account while designing security test cases. The relevant parts for test cases within the standard are shown in figure 4.2. This figure shows three parts of the testing standard, which are incorporated into the later described structure of the functional security test cases. The standard is used as reference to create a structure for the cases which is based on international agreed assumptions. As explained in [KFN99], the standard can not be seen as an exact description of how a testing approach has to look like, but as a framework for creating adapted testing approaches for individual purpose. According to figure 4.2, the *Test design Specification* sums up a test case from a top level view. This specification can include multiple test cases which are used to test a certain feature. These test cases are represented by the *Test case Specification*. It represents a test case with all its inputs, outputs, environmental needs and other attributes, which are further described in [ISO98] and later

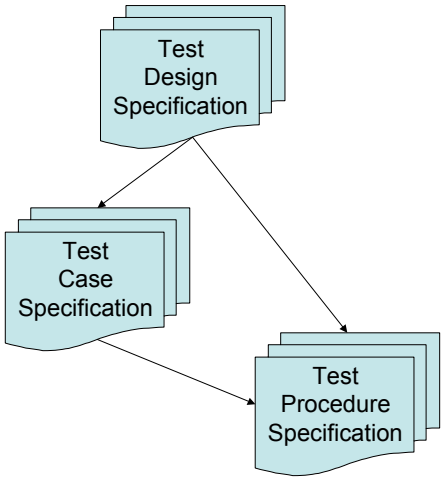

Fig. 4.2 Extract of the testing standard from [ISO98]

during this work in section 4.4.The *Test procedure Specification* contains procedural instructions about the test case and how it should be conducted. In this work these components are used and adapted for the design of the database schema. It has to be mentioned that only the underlying ideas are utilized rather than the standard is strictly implemented. This is caused due to its holistic view and the need for adaption to functional security testing and *TeliaSonera*. As we will show in this chapters the structure of the even presented three documents is not kept straight forward due to conceptual design reasons and the way the artifacts are used.

4.4 A database for Functional Security Testing

To manage and store a huge number of security test cases it is obvious that a database fits to this need. The purpose of such a database is subsequently described:

> The aim is to capture security knowledge to support the testing process as described in section 4.2.

This can be seen as the main idea and driving force for such a database. It targets on supporting the security testing process of *TeliaSonera*. An useful and sophisticated database schema for this task must fulfill the following requirements. These requirements are mainly collected through interviews

with my supervisor at *TeliaSonera*, *Albin Zuccato*, and best practice information from [KFN99].

- Mapping between top level security categories and their related security mechanisms
- Mapping between security mechanisms and their related requirements
- Mapping between requirements and their related test cases
- Different security levels must exist for different protection needs according to the CIA criteria [Gol06], [PP03].
- It must be possible to determine the protection level of requirements and their related test cases according to the CIA criteria.
- It must be possible to extract related test cases of a requirement for specific security levels.
- Regression levels must be included, which determine how important and critical certain test cases are.
- It must be possible to get the related test cases of a requirement according to a given regression level
- Test cases can be dependent on other test cases which must be executed in advance
- Facts about conducted test cases and their success or failure must be available for statistical evaluation
- The schema must be flexible to provide long-time usage

4.4.1 Identification of entities

Due to these requirements a couple of necessary entities can be extracted. The following top level entities are identified, but it has to be mentioned that they do not take any special design methods into account yet.

Category The *Category* is used to categorize security mechanisms according to a global view of security. Such a category can be for example *Authentication*.

Mechanism *Mechanism* is a more concrete security mechanism which is directly linked to a requirement. The category and the mechanism entities also occur in the SMRT as explained in section 4.1.

Requirement The *Requirement* refers to a system requirement, which can be tested during the testing activities within the development life cycle. A requirement can be on the one hand very specific and concrete. On the other hand it has to be mentioned that requirements, which are only described by a top level specification, occur as well.

Test Case A *Test Case* represents a functional security test case. The aim of a test case is to capture specialized security expertise to make it available if needed. As explained above, a mapping between a requirement and its related test cases should be enabled by the presented approach.

Security Level To determine which test cases are relevant to perform is not possible without knowing which *Security Level* a certain requirement must fulfill. The security level entity provides these security levels, to which a requirement and a test case can refer to. This determines on which security level a requirement is tested by a test case.

Regression Level The use of *Regression Levels* creates the opportunity to assign test cases to different classes. These classes can be differentiated by their importance. Hence this classification can be used to decide which tests are more important than others. When it comes to projects which suffer from a high time and cost pressure, this can be used to prioritize tests. Further this aims on an utilization in regression tests according to section 3.4 and should be seen as an extra feature which is directly built in in the approach even though it is not the primary focus of the work.

Conducted Tests The last entity is called *Conducted Tests* and provides an opportunity for documenting the performed activities and their results. Further it builds the foundation for later analyzations. The collected data can be evaluated to get statistical information about how many failures the test cases identify, which areas are affected by a larger number of security problems, or other important statistical evaluation criteria.

4.4.2 Top level schema

Due to the complexity of the relations between the identified top level entities, the mapping of the conceptual functional security testing approach to a database schema is not straight-forwarded and offers multiple opportunities. One possible database schema is shown in figure 4.3. This schema shows only the relation between the described top level entities and does not include attributes or information about how relations will be implemented within the actual database. The concrete entities including their attributes and the technical implementation are described later in this section. The presented schema is designed with the aim to create a highly flexible and adaptable database. This is caused by the above described requirements. The top level schema does not pay attention to performance issues due to the estimated small size of the database which would not justify to put effort in such optimization. It can be said that current systems have enough resources to handle it.

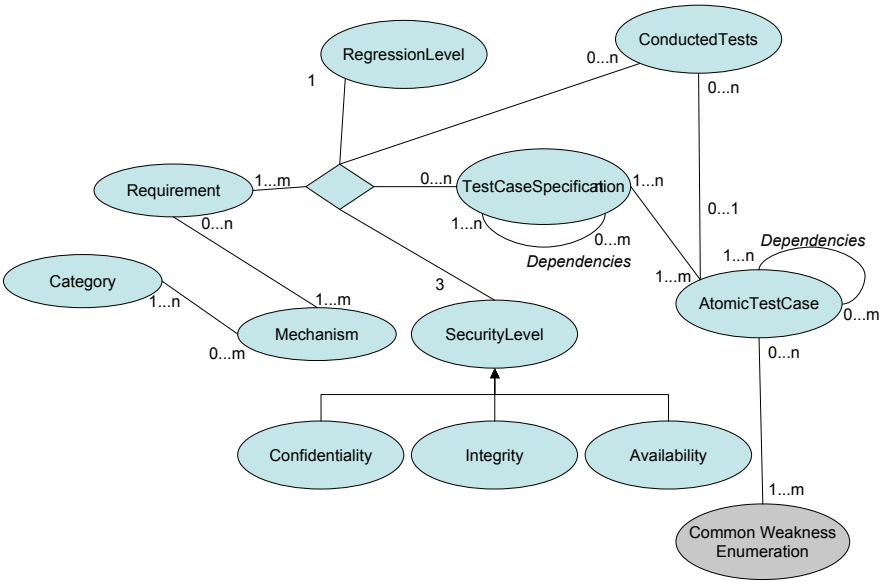

Fig. 4.3 Top level database schema for the functional security testing approach

As it is shown in figure 4.3 the *Category* entity can have multiple related *Mechanisms* and a mechanism can also belong to multiple categories. This causes the relation between them to be stated as *n:m*.

The relation between a *Requirement* and a *Mechanism* is identified as *n:m*. A mechanism has multiple requirements. While this is quite obvious, it is also possible that a requirement belongs to multiple mechanisms. This can occur when a requirement cannot be clearly assigned to a certain security task, or when mechanisms are interrelated.

The *SecurityLevel* can be split up in three subtypes where every type can determine either the level of confidentiality, integrity or availability. This distinction between the three different security perspectives is necessary due to the above described requirements and the need to specify a requirement according to these security level criteria. Hence a requirement has a *n:3* relation to the security level.

The *TestCaseSpecification* entity represents a test case. The relation between this entity and the requirement entity is *n:m* as a requirement needs often more than one test case to be tested and a test case can be used in more than one specific requirement. The test case specification itself can have other, multiple test case specifications as dependencies. This leads to the recursive dependency relation of the test case specification which is *n:m*. As a test case specification can have multiple preconditions and a test case

specification, which is used as a precondition, can in turn be used by multiple other test case specifications, it is necessary to have such a relation.

The *AtomicTestCase* entity is used to create a hierarchy within the test cases. Because it is useful to reuse parts of a test case to avoid redundancy this hierarchical structure is introduced. Due to that a test case specification can take multiple atomic test cases to form a test case and a certain atomic test case can be used by multiple test case specifications. This leads to the usage of an *n:m* relation. The structure has been chosen with the aim to create a flexible design as explained earlier. It is also used to avoid redundancy and hence lead to a better database design. An atomic test case has a recursive *n:m* relation, which is caused due to the need for dependencies. The reason for that is equal to the above described one for the test case specification.

The *RegressionLevel* specifies the importance of a test case during a regression test. As a test case under the influence of security levels and requirements can only have one certain regression level and a regression level is used by multiple test cases a *1:n* relation is used.

The central point of the schema is the relation between *Requirement*, *Security Level*, *TestCaseSpecification*, *RegressionLevel* and *ConductedTests*. This relation maps a requirement to its test cases, where every linkage between them has its specific regression level and security levels according to the CIA criteria. This relation is unique. Hence one test case specification can test multiple requirements, where every requirement can be tested on a different security- and regression- level.

The *ConductedTests* entity collects statistical information about performed tests, which can be used for a later analyzation. It has no influence on any other relations. It is used to collect information about all performed test cases and their results. It refers to atomic test cases as well as to the even described multiple relationship to provide traceability for every executable part.

An additional entity, which is not derived through the given requirements and also not further mentioned for the database design is the *Common Weakness Enumeration* (CWE). The CWE is a collection of software weaknesses which can be found in [MIT08b]. The integration of the entity enables a link between a test case and a weakness it tests for, thus creating a more holistic view of the coverage of known weaknesses by test cases. As this is outside the scope of this work, this entity is not further used for the derivation of the relational- and database- schema. However, it is mentioned as proposal for a further development of the functional security testing approach.

As described there are many *n:m* relations between entities, which is caused due to design aim for flexibility and adaptability, but also due to the given complexity of the relations. This flexibility causes a responsibility shift for the meaningfulness of the inserted content from the schema to the application or user. On the one hand this can be seen as a tradeoff for

flexibility. On the other hand it offers a whole range of opportunities to use it, rather than a strict database design. It has to be mentioned that data logic and constraints are later used in this chapter to provide consistent data while still keeping the flexibility high.

In reference to [ISO98] the presented database schema slightly changes the proposed test case structure. However, it keeps the basic ideas. The *Test design Specification* is renamed to *TestCaseSpecification*. The *Test case Specification* and the *Test procedure Specification* are merged and represented by the *AtomicTestCase*. This merge is done due to simplification reasons and is caused by the fact that no benefit is generated for this field of application by the separation of these artifacts.

The presented top level schema gives an overview of how the test cases and requirements are organized within the database. The principal structure of the approach is described and according to that it is now possible to refine it. This more detailed observation changes the focus from a generic top-level view to the attributes and relational implementation of this structure.

4.4.3 Relational schema

According to the above described top level schema of the database it is possible to refine it and thus create a structure which can be directly mapped to a relational database. This could be done by a simple textual relational schema. To make it easier to understand, figure 4.4.3 presents it in a graphical way, which is now further explained in this part. The graphical representation makes the structure and its related components more obvious than a classical textual relational schema would do. Another advantage is that it contains more information than a classical relational schema, as for example the used data types for attributes.

The *Category* table consists of a name and a description. Additional it has a primary key, which is stated out by bold letters and uniquely identifies each category. As every entity contains an unique identifier, this is not explicitly mentioned for every table.The *Mechanism* table is similar to the category table and contains an unique identifier and a name. To represent the mapping between both tables, the *Cat2Mech* table is utilized. The *Requirement* table includes a name and a description. The mapping between a category and a requirement is represented by the *Cat2Req* table, which allows a multiple (n:m) mapping between requirements and categories. It has to be mentioned that all of these three hierarchy levels have a multiple (n:m) mapping. This is caused by the requirement to make the relations as flexible as possible.

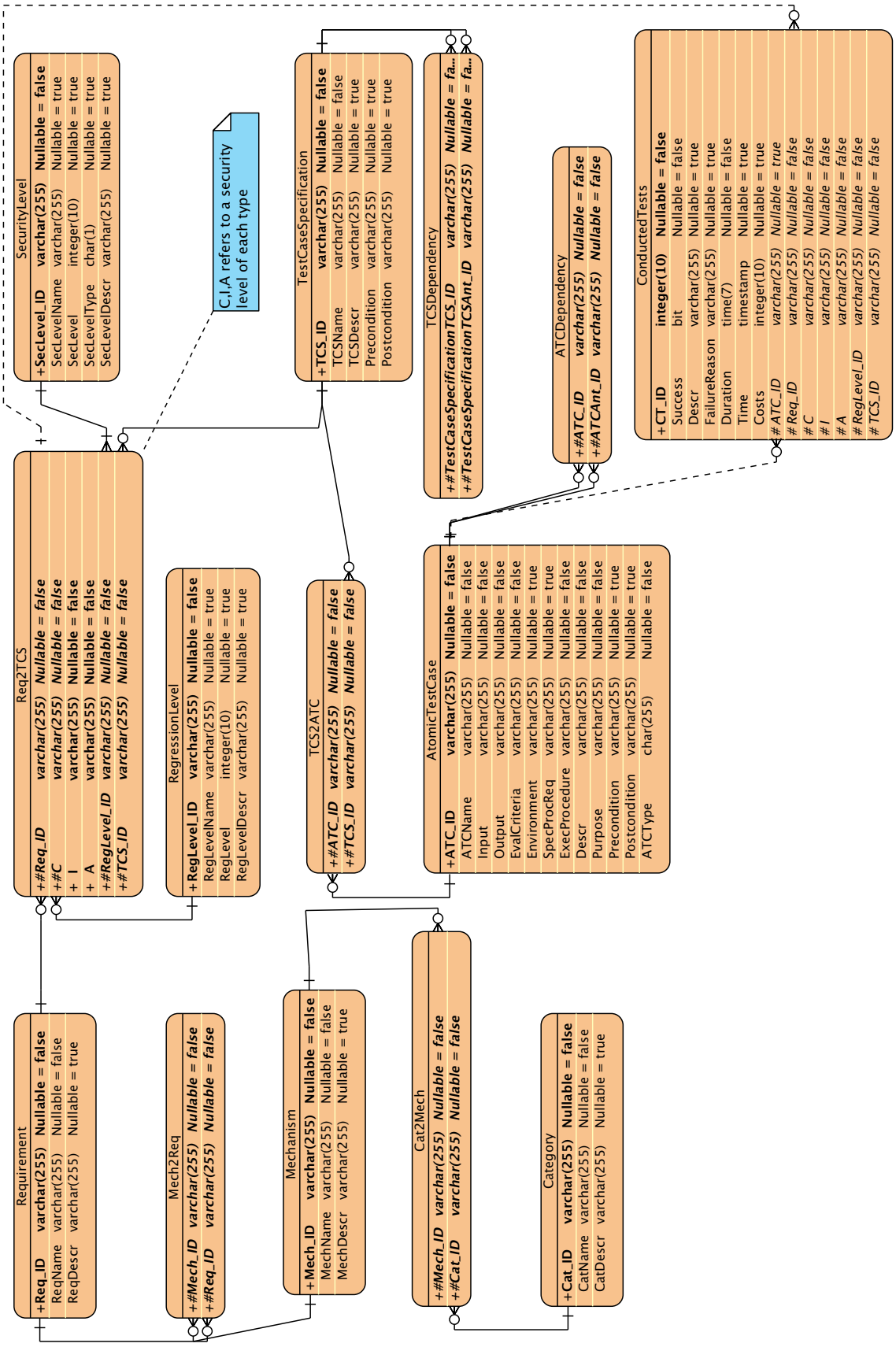

Fig. 4.4 Relational schema for the functional security testing approach

The *SecurityLevel* table includes a name, which gives a hunch of the meaning of the security level. In addition it has a description, which is used to further explain details. The *SecLevel* field is necessary to provide a ranking between security levels according to their strength. This ranking offers the opportunity to compare security levels and hence use them according to the given security needs to find matching test cases. The table also includes a type specification, which is utilized to state if a security level belongs to the C, I or A class. As it can be seen when comparing the top level schema in figure 4.3 with this relational one, the hierarchy of the security levels is not transformed into a own table for each type and subtype. This is caused due to the fact that the estimated number of security levels will be small and no advantage is given by the use of an explicit representation of the hierarchy. Hence the type categorization is done by the *SecLevelType* attribute. This attribute contains either an C, I or A to specify the type of the security level.

The *RegressionLevel* table is similar to the even described security level. It includes a name and a description as well as a *RegLevel* attribute. This attribute is used to rank between different regression levels. As no hierarchical representation is necessary, no type attribute is needed in the regression level table.

The *TestCaseSpecification* table includes a name and a description. Even tough this table presents a test case it does not contain the necessary information for the testing itself. It uses the *AtomicTestCase* table to keep this information, while the specification table itself aggregates this atomic test cases to new test cases. Due to the possible interdependency of a test case specification to another test case specification the *TCSDependency* table is used to provide a (n:m) mapping between them. This provides that a test case specification can have multiple antecessor which have to be conducted in advance and still can be used as antecessor for other test case specifications. The attributes *Precondition* and *Postcondition* refer to actions which have to be performed or states that have to be reached before using a test case specification and further conducting its atomic test cases. These attributes include no specific pre- and post conditions for the linked atomic test cases, but they are used for more general aspects. This can be the need for user accounts on the evaluated target system with special privileges or special software tools which are necessary for a proper executing of the tests. However, an atomic test case itself has further the same attributes for pre- and post- conditions, but they have to be used differently as further described in this section.

The mapping between a requirement and its related test cases is shown by the *Req2TCS* table, which represents the multiple (n:m) relationship. Due to that a test case specification can be reused by different requirements and a requirement can include multiple test case specifications. This relation is not only determined by the requirement- and test case specification table.

As it can be seen the security- and regression level plays an important role when determining the relation between a requirement and its test case specifications. The *Req2TCS* table includes attributes which refer to a regression level and different types of security levels. The *RegLevel_ID* refers to the regression level the requirement should be tested on by the determined test case specification. The *C*, *I* and *A* attributes refer to a security level of a C, I and A type, which was explained above. The relation between a requirement and its test case specifications can be described as followed:

> A requirement can have multiple test case specifications which test the requirement on different security- and regression levels. Hence a test case specification can test different requirements on different security- and regression levels.

When regarding this table from a more technical point of view an important point is the primary key. The relation between a requirement and its test case specification is further determined by the security- and regression levels. Hence this unique combination of a requirement, test case specification, security- (C, I and A) and regression level is directly taken as a primary key. Summarized it can be said that the relation between a requirement and its test case specifications is complex and must be further determined by security- and regression levels in order to achieve a correct and useful mapping between them.

An *AtomicTestCase* is a basic test case. This basic component is separated from the real test case, which is represented by the test case specification. This separation is caused due to reusability reasons and to avoid redundancy within the database. The atomic test case includes the following fields, which are taken into account according to [ISO98].

- ATC_ID
- Name
- Input
- Output
- EvalCriteria
- Environment
- SpecProcReq
- ExecProcedure
- Descr
- Purpose
- Precondition
- Postcondition
- ATCType

The *ATC_ID* is a unique identifier which is not used to further specify the atomic test case for humans, but for the primary key of the table. The *Name* gives a short description of the test case. This is used to support humans when creating new test case specifications from given atomic test cases and to give a short overview of what the test case component is about.

The *Input* field specifies the necessary input to conduct the test case. It has to be mentioned that this input must not be a list of concrete data sets, but can also be an accurate description of how the input must look like. This is caused due to a higher flexibility and a wide range of different inputs, which can not all be treated explicitly for each system and requirement that will be tested.

According to the input the *Output* is specified. This is done similar to the input creation. The *EvalCriteria* attribute contains criteria which are used to determine if the conducted test was successful. The content of this field can be, according to the input and the output, either an accurate top level description of the pass- or fail condition, or an exact listing of the criteria a test has to fulfill to be counted as successful or failed.

In the *Environment* attribute environmental needs can be described. These requirements include any environmental precondition or a certain environmental affected state in which a system has to be while conducting the test. It is important to notice that other atomic test cases, which have to be performed in advance, are not included in this field. They are represented by an explicit table for this kind of precondition, which is later described during this part. The *SpecProcReq* contains special requirements for the procedure of performing the test case. For further explanation of these special procedural requirements see [ISO98].

The *Descr* attribute contains a description of the test case. The aim of a test case is stored in the *Purpose* data field. It gives a short explanation of what the test case is testing for. The *Precondition* field is used to describe preconditions for the test case. These preconditions are in comparison to the *Environment* field not environmental. They aim at certain system or component specific actions or prerequisite which have to be done or fulfilled before conducting the test. The *Postcondition* refers to actions which have to be performed after the test case was conducted. This can be for example a cleanup activity.

To further specify for what kind of test an atomic test case stands for, the *ATCType* attribute is used. It refers to the different functional security test techniques presented in section 3.3. Hence this attribute can take only several states:

- Requirement
- Equivalence

- Boundary
- Robustness
- Usage-based

This additional classification can be seen as an extra feature. Its is especially useful for later statistical analysis of conducted test cases and the assessment of a requirements coverage by its test cases as later described in chapter 6.

The above mentioned fact, that an atomic test case can have multiple atomic test cases as antecessors and such a case can also be used as precondition by multiple other cases, leads to the need for a mapping table. This mapping table is represented by *ATCDependency*, which is utilized to manage these relations. A remaining part to describe in the model of figure 4.4.3 is the relation between the test case specification and the atomic test cases. The *TCS2ATC* table maps multiple test case specifications to multiple atomic test cases (n:m). Hence it allows the reuse of atomic test cases by different test case specifications.

The last entity is the *ConductedTests* table. It stores statistical information about the performed tests and their results. In order to achieve a flexible possibility to trace information not only about the conducted atomic test cases, test case specifications or the tested requirements, it has relations to all these relevant tables. This provides to store all information in just one table. Hence complexity is reduced. As it is an extra table for additional information storage, it has no influence on the structure of the presented database schema itself, even if it has multiple relations to its tables. The attributes are not further explained here as their names are a sufficient explanation.

During this description no attention has been payed to the used data types, as they can vary between different relational databases. The used data types should only be seen as hint to the kind of content for the attributes, not as strict limitation. The additional information about the ability to leave attributes empty is also only a very basic one and presents just an expedient assumption. These recommendations are then refined and set out in the implementation part. As it can be seen when regarding the holistic structure of the presented database schema, generalization is widely used. It is utilized to craft a logical and usable structure, which leads to a more comprehensible build-up. The core component is the *Req2TCS* table which includes all relevant data for structuring the relationship between requirements and its corresponding test cases. Summarized it can be said that this mapping table directly refers to the aim of this thesis as it is stated out in chapter 1:

> It is possible to create a correlation between functional security tests and given security requirements, which enables a direct mapping of test cases to their underlying requirements.

4.4.4 Implementation

During this part the presented relational schema from subsection 4.4.3 is transformed into executable SQL code. As there are different SQL coding standards it has to be mentioned that the target system for the here presented code is MySQL [MA07]. Following important decisions and assumptions for the derivation of the SQL code through the relational schema are presented. Simple and obvious transformation details are left apart to reduce redundancy.

The *Req2TCS* table as shown in listing 4.1 represents the core mapping between requirements and its test case specifications.

Listing 4.1 Mapping between Requirements and Test Case Specifications

```
1  CREATE TABLE Req2TCS (
2          Req_ID              VARCHAR(255) NOT NULL,
3          TCS_ID              VARCHAR(255) NOT NULL,
4          C                   VARCHAR(255) NOT NULL,
5          I                   VARCHAR(255) NOT NULL,
6          A                   VARCHAR(255) NOT NULL,
7          RegLevel_ID         VARCHAR(255) NOT NULL,
8          PRIMARY KEY(Req_ID, TCS_ID, C, I, A, RegLevel_ID),
9          CONSTRAINT Req2TCS_Req_dependency FOREIGN KEY (Req_ID)
10                 REFERENCES Requirement (Req_ID)
11                 ON UPDATE CASCADE
12                 ON DELETE CASCADE,
13         CONSTRAINT Req2TCS_TCS_dependency FOREIGN KEY (TCS_ID)
14                 REFERENCES TestCaseSpecification (TCS_ID)
15                 ON UPDATE CASCADE
16                 ON DELETE CASCADE,
17         CONSTRAINT Req2TCS_Reg_dependency FOREIGN KEY (RegLevel_ID)
18                 REFERENCES RegressionLevel (RegLevel_ID)
19                 ON UPDATE CASCADE
20                 ON DELETE CASCADE,
21         CONSTRAINT Req2TCS_C_dependency FOREIGN KEY (C)
22                 REFERENCES SecurityLevel (SecLevel_ID)
23                 ON UPDATE CASCADE
24                 ON DELETE CASCADE,
25         CONSTRAINT Req2TCS_I_dependency FOREIGN KEY (I)
26                 REFERENCES SecurityLevel (SecLevel_ID)
27                 ON UPDATE CASCADE
28                 ON DELETE CASCADE,
29         CONSTRAINT Req2TCS_A_dependency FOREIGN KEY (A)
```

```
30                      REFERENCES SecurityLevel (SecLevel_ID)
31                      ON UPDATE CASCADE
32                      ON DELETE CASCADE
33 )ENGINE=INNODB;
```

To avoid failures when creating relationships in this table, constraints are used to protect the data integrity. As it can be seen in line *8*, the primary key is created by the use of all attributes, which was already described in the relational schema. For each of the attributes a constraint to its original table is set (line *9, 13, 17, 21, 25, 29*). This assures that a change in any key value does not lead to a breakup of the mapping table. Another important benefit from the constraints is that any kind of insert, update or delete inconsistency can be avoided due to their use. This data consistency logic supports the later insertion and management of data. Hence it leads to an easier and more efficient usage during the life cycle.

To store atomic test cases the table in listing 4.2 is used. As it can be seen not every attribute is mandatory. This enables a more flexible usage of the given structure, which also provides special atomic test cases to be stored within the database. This helps to reduce the need for workaround procedures due to future changes. The *ATCType* attribute in line *14* refers to the functional testing techniques described in section 3.3. Hence the possible input values are limited to the presented methods. This helps on the one hand to avoid wrong, in a sense of not included or incorrect, classification of a test case. On the other hand it prohibits inconsistency due to the avoidance of different spelling of the same type. As this could be seen as contradiction of the aim to create a flexible structure, it has to be mentioned that this can be easily changed and extended afterwards without affecting any other elements of the database.

Listing 4.2 Atomic Test Case

```
1  CREATE TABLE AtomicTestCase (
2          ATC_ID  VARCHAR(255) PRIMARY KEY,
3          ATCName VARCHAR(255) NOT NULL,
4          Input           TEXT,
5          Output          TEXT,
6          EvalCriteria    TEXT NOT NULL,
7          Environment     TEXT,
8          SpecProcReq     TEXT,
9          ExecProcedure   TEXT NOT NULL,
10         Precondition    TEXT,
11         Postcondition   TEXT,
12         Descr           TEXT NOT NULL,
13         Purpose         TEXT NOT NULL,
```

```
14          ATCType ENUM("boundary","requirement","robustness",
15                  "usage-based","equivalence") NOT NULL
16  )ENGINE=INNODB;
```

The above described tables do not represent the whole database schema, which can be found in appendix A. The missing tables, which are not further described, are created through simple derivation from the relational schema. It has to be mentioned that during the development of the sql schema attention has not only been paid to flexibility, but on the use of constraints and due to that the use of data logic. This provides a flexible but nevertheless consistent database. Hence it enables to meet not only current specifications, but builds a foundation to react to changing requirements in the future.

4.5 Creating content for the functional security testing approach

To generate value for *TeliaSonera* due to the functional security testing approach not only its usage, but also its enrichment must be documented and integrated in the organizational structures. The enrichment is in this case seen as the enlargement of the approach through the addition of new content. This widens the applicability and received benefit. It does not refer to an extension of the ideas of the approach itself. To document the development of new content for the approach, a process fits best to capture all included activities, artifacts and environmental needs. Figure 4.5 shows this process.

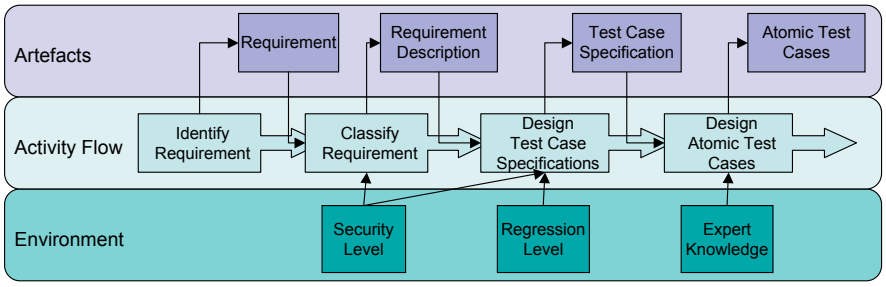

Fig. 4.5 Process for the creation of content for the functional security testing approach

Requirement identification and classification At the beginning of the process a new requirement is identified. This requirement is then further described to provide an integration in the given structure of categories and

mechanisms. If the requirement cannot be added properly into the given structure, a best-match approach should be used. This may seem ambiguous, but it has to be mentioned that the process refers to the usage within *TeliaSonera*. A change of the static structure would cause a number of changes in other processes. However, it has to be mentioned that the given static structure covers a wide range of the security area and the necessity for adaption is of low probability. The classification of the requirement does not only refer to the integration in the static structure, but also to a classification according to the given security levels. Even though the security levels are not determined until the creation of the test case specifications, they should be taken into account at this step of the process to get a presentiment about the level of detail the further test cases must cover the requirement and the maximum achievable security level.

Design test case specifications The test case specifications represent the aggregated atomic test cases. As explained in this chapter, they determine the security levels a requirement can be tested on. Hence they form the core component to constitute which protection level a requirement provides. As tests can give a hunch about the correctness of a requirement and untested behavior, which includes untested security levels, cannot be seen as assured, they determine the achievable security level of a requirement. *Regression Levels* are an additional environmental factor which affects the design of test case specifications. The specifications include information about their importance for regression tests, which must be set in correlation to the given regression levels.

Design atomic test cases Atomic test cases form the executable tests. When designing such tests, expertise about the requirement must be taken into account. Due to that the environmental factor *Expert Knowledge* influences this activity. Further the functional testing techniques from section 3.3 are used when creating the tests. A more detailed explanation of crafting atomic test cases in relation to real requirements is given in section 5.2.

Another important point which has to be mentioned is the reuse of test case specifications and atomic test cases. When performing the process for a new requirement, the reuse of already existing components is desirable. Through the reuse of existing test case specifications and atomic test cases a faster and thus more cost efficient usage of the approach can be achieved. Not only the reuse of test case specifications for a new requirement should be taken into account, but also the reuse of already existing atomic test cases to aggregate new test case specifications. This enables a reduction of the amount of test cases, which increases the simplicity while keeping the generated value by the use of the approach on the same level.

Summarized it can be said that the process shown in figure 4.5 is a guide to enlarge the content base for the functional security testing approach. It describes top level activities, as well as the generated artifacts and the relation to environmental factors which must be taken into account. Thereby the aim of the process is not to restrict the developer during the creation of new content, but to provide a generic guideline. In section 5.2 the usage of the process is shown to describe the creation of new content according to example requirements.

Chapter 5
Derivation of Functional Security Test Cases

The derivation of functional security test cases is a task which does not only include knowledge about the approach from chapter 4. It further requires security knowledge. Thus both factors must be taken into account to perform it sufficiently. When crafting functional security test cases it is beneficial to keep the testing techniques mentioned in section 3.3 in mind. According to these techniques it is possible to create various functional security test cases. In addition the crafting process is influenced by some environmental factors, which reach from given structural perquisites by *TeliaSonera* till common security issues.

In this chapter detailed insight into the creation of content for the developed functional security testing approach is given. At the beginning the influence of the *Security Mechanism Reference Table* (SMRT, see section 4.1) on the creation of content for the database from chapter 4 is shown. Further the process for the creation of new content for the functional security approach from section 4.5 is applied. Two representative requirements are taken to show the crafting of test cases according to the process.

5.1 Predefined Structure

It is useful to build-up on a basic structure of categories and their related security mechanisms, when creating content for the database shown in chapter 4. As this work is related to the processes and methodologies used by *TeliaSonera*, the basic structure must follow them. Due to that the SMRT, as explained in section 4.1, is taken into account. It already had influence on the design of the database, but also provides basic content. Hence the following categorization for security mechanisms is mandatory:

- Authentication

- Access Control
- Data integrity transfer protection
- Data confidentiality transfer protection
- Stored data integrity
- Traceability
- Restoreability
- Key Management
- ...

As already mentioned in section 4.1, each of the categories offers different protection according to the C, I and A criteria. This is not further mentioned within the static structure of the category, as it is individually determined for each requirement and its test cases (see chapter 4). Thus enabling a more flexible and detailed description with attention to different protection levels provided by different requirements. These protection- or security- levels are also predefined due to the SMRT:

- Level 0 : No protection
- Level 1 : Low/Basic protection
- Level 2 : Medium protection
- Level 3 : High protection

It has to be mentioned that *Level 0* is not a real level as it only refers to the absence of any protection mechanism. Not only the use of no security mechanisms, but also a missing verification of the functionality of implemented security mechanisms leads to a classification on this level. Due to that it is not possible to create tests for a mechanism or requirement on *Level 0*. The other security levels represent increasing levels of protection according to their increasing numbers. When protection on a certain security level is required, a requirement which provides protection on this level has to be implemented. To assure that a certain implemented requirement fulfills its specification, functional security tests are used.

Further the SMRT describes mechanisms, which can be used to achieve different security levels. However, it leaves the concrete implementation and the used security requirement to the user. This provides a flexible usage of the SMRT in different fields of application. The different mechanisms are not explained here in detail, as it would go beyond the scope of this work.

Summarized it can be said that due to the use of the SMRT some content of the database is predefined. This basic structure contains categories for security mechanisms and protection levels. They are used to create a structure within the functional security testing database to provide a fit to the processes of *TeliaSonera*.

5.1.1 Regression Levels

The *Regression Levels* are an extra feature of the approach. Due to the fact that the use of such regression levels generates benefit by nearly no extra costs, they are incorporated in this work. The aim of regression levels is to improve testing in case of limited resources. When not enough resources can be allocated for conducting all necessary testing activities, regression levels provide a classification which allows to perform at least the most critical test cases. Without a classification of importance, it is complex to make such decisions. As no given regression levels exist in *TeliaSonera*, we created them according to the already presented security levels. The following listing shows the defined regression levels:

- Level 0 : No classification
- Level 1 : Low importance
- Level 2 : Medium importance
- Level 3 : High importance

As it can be seen, the regression levels are divided into three stages. Similar to the described security levels in section 5.1, *Level 0* does not actually exist. The classification of a test on this level indicates that there is no actual classification and the regression level feature is not used. *Level 1* represents the lowest level of importance. Hence tests on this level can be skipped without missing to check critical functionality. *Level 2* represents the medium level of importance and includes more critical tests than *Level 1*. *Level 3* includes the most critical tests which should be out of question to perform. These mandatory tests include critical system functionality. Summarized it can be said that this classification of tests is not only capable of facilitating decision making in case of limited resources. Further it can be used for regression tests, which are described in section 3.4.

5.2 Crafting of Functional Security Test Cases

According to the above mentioned basic structure, security mechanisms and requirements can be identified. A mechanism is a general description of security functionality, while a security requirement can be specific and designed for the individual need of a system. Due to the diversity of systems and their individual needs it is nearly impossible to list all existing security requirements or mechanisms used by *TeliaSonera*. However, it is possible to cover the most common, which provides a base for creating test cases.

When starting to craft functional security test cases, the mentioned functional security testing techniques play a major role. Expertise about the used requirement and its functionality is indispensable for creating useful test cases and the use of different functional security test techniques can be seen as an indication for the coverage of a requirement by them. Hence these techniques provide support for the creation of test cases. A structured overview of identified mechanisms, requirements and their related test cases can be found in appendix B. This list is an extract of the content of the database, which was presented in chapter 4. It has to be mentioned that this list does not include most of the developed content, as this is restricted information for *TeliaSonera*, which cannot be published. However, it gives a hunch of the content and its structure.

Further it has to be mentioned that at this early stage only a coverage for common requirements is included in the database. This is also caused due to the fact that the main focus of this work is on the development and implementation of the approach, rather than a full coverage of a large amount of requirements used by *TeliaSonera*.

Due to the number of different test cases and their requirements, a detailed explanation of all of them would be beyond the scope of this work. Hence two examples are now presented in detail. Thereby the focus lies not on a precise explanation of implementation details, but on the process for crafting test cases to cover the specification of a requirement (see section 4.5). The chosen examples represent two different kinds of requirements. The one presented in subsection 5.2.1 (*Standard One Factor Authentication*) shows a generic requirement which does not include many technical details about the actual implementation. The other one in subsection 5.2.2 (*SSL/TLS*) is specified by a strict technical standard. This diverse orientation of requirements has an impact on the possibilities to test for their functional correctness. Hence the test cases must pay attention to this different focus of the requirements.

A difference between generic requirements and specific requirements is the relation to their mechanism. As the mechanism is on the above hierarchy level, it can contain multiple requirements. In case of a generic requirement the mechanism often contains only one. Hence one generic requirement can represent a whole mechanism. While the separation between requirements and mechanism seems to lead to no benefit for generic requirements, it is indispensable when it comes to specific requirements. Specific requirements can be aggregated by their mechanisms, which is an necessity in regard to the SMRT and the mapping between requirements and their test cases as described in chapter 4.

Even tough it would be possible to create only specific- or generic requirements, this would lead to an enormous number of test cases on the one

hand, or an inapplicability of the test cases by the testers on the other. When only specific requirements would be used, the number of requirements in *TeliaSonera* would grow to an unnecessary high amount without generating benefit. This is caused due to the different implementation details, which must be taken into account. In case of the requirement in subsection 5.2.1 (*Standard One Factor Authentication*), a higher grade of detail can be seen as unnecessary. This is further explained in this chapter. On the other hand specific requirements can be indispensable and useful when the implemented security functionality of a mechanism must follow a strict technical specification. To protect transfered data over a network a VPN solution like *IPSec* [IET98] or SSL/TLS [IET99] (subsection 5.2.2), which are both exact specified, can be used. As they have different approaches to achieve protection of the confidentiality and integrity of transfered data, a generic requirement which covers both would not sufficiently fulfill the need of supporting *TeliaSonera's* testing process.

Summarized it can be said that requirements can differ from generic top level- till to strict technical ones. Both are covered by the developed functional security testing approach.

5.2.1 Example A - Standard One Factor Authentication

Table 5.1 shows a requirement which is part of the *User Authentication* category. The related mechanism is nearly the same as the requirement, as it can be seen as a generic requirement which has no need for a stronger technical orientation. This is further explained in this subsection.

Name	Description
Standard one factor authentication	Use of username, id, costumer number etc. and password, pin, etc.

Table 5.1 ExampleA Requirement

As it can be seen in table 5.1, the requirement is not specific. This is caused due to the common usage of such a requirement in many systems, where only minor functional differentiation criteria can be identified, but the actual implementation still differs. Hence such a generic requirement can be used in this case. In other cases, as for example the protection of data during the transfer over a network, requirements can be more specific. This is further discussed in the next subsection 5.2.2.

5.2.1.1 Test Case Specifications

According to the given requirement in table 5.1, it is possible to create related test cases. Regarding the presented approach from chapter 4 the test cases are divided into the *TestCaseSpecification* and the *AtomicTestCase*. The specification aggregates atomic test cases, while the atomic test cases represent the executable tests. Atomic test cases contain only as much information, as there cannot be an overlap between them. It can be said that atomic test cases represent atomic and mutual exclusive parts of tests, which are aggregated by the test case specification to create tests for a certain requirement.

In this case it should be possible to test the requirement on different regression levels. Hence the following test case specifications can be identified:

Name	Description
Basic one factor authentication	Tests if the one factor (e.g. password) authentication works properly on a basic level
Advanced one factor authentication	Tests if the one factor (e.g. password) authentication works properly on an advanced level

Table 5.2 Test Case Specifications for table 5.1

As shown in table 5.2 the requirement from table 5.1 is linked to two test case specifications. Both aggregate atomic test cases, which is further explained during this section. Attention has to be paid to security- and regression levels at this point as they influence the test case specifications. As the influence of the security level seems to be obvious, it is important that the regression level is taken into account as well. Regression levels classify the importance of a test case specification. Hence it is necessary to keep this in mind in order to further aggregate only atomic test cases of the same importance in a test case specification. In this particular case the split into two test case specifications is caused by the need to test the requirement on different regression levels. An important point to mention is that the test cases should not be too extensive or complicated, as their execution is mostly delegated to non-security experts. Hence a balance between the sophistication and the applicability of the test cases must be kept. In this case the two test case specifications for the requirement show a coverage on a basic- and advanced level. Such a separation is influenced by the need for reusable test case specifications, as they may not be used together when testing the same require-

ment on different regression levels. The security classification according to the given security levels is explained later in this part.

5.2.1.2 Atomic Test Cases

When creating atomic test cases for the given requirements the functional test techniques and expert knowledge should be taken into account. While the test techniques create a framework in which the test cases should fit in, the expert knowledge is indispensable for creating test cases within this area. Additional common software problems should be targeted during the creation of test cases. A collection and categorization of such weaknesses and vulnerabilities can be found in the *Common Weakness Enumeration (CWE)* [MIT08b] or *Open Web Application Security Project (OWASP)*[OWA08]. This builds a good source for the kind of required knowledge if no experts are available. Thereby it has to be mentioned that these sources are used without a malicious intend. The aim is to find common problems of similar systems, not to use this knowledge to break it. According to [Gol06], [VM02], [KFN99] and [MIT08b] the following atomic test cases can be identified:

Attribute	Value
ATC_ID	ATC0
ATCName	Generic one factor authentication test
Input	Correct Username. Wrong Username. Correct Secret. Wrong Secret
Output	Login successful or forbidden
Evaluation Criteria	Login was only successful with correct username and correct secret. Meaningful error messages are given. Responds time for successful and unsuccessful logins must be equal. Transmission during login must be encrypted
Environmental requirements	Logging is enables to verify system messages. Valid account exists and is usable. Timer that can measure reaction time of system.
Special Procedural Requirements	Check error Message for unsuccessful login attempts. Check responds time for successful versus unsuccessful (all types) login. Record transmission (if exists).
Execution Procedure	Login(correct username, correct secret). Login(wrong username, correct secret). Login(correct username, wrong secret. Login(wrong username, wrong secret).
Precondition	

Postcondition	
Description	This test assures that the authentication mechanism works correct on a basic level and only allows correct authorized users to access the system
Purpose	Test the one factor authentication on a basic level
Type	usage-based

Table 5.3: Generic one factor authentication test

Attribute	Value
ATC_ID	ATC1
ATCName	Secret change
Input	Correct Username. Correct secret. Low quality secret. High quality secret
Output	Login successful or forbidden
Evaluation Criteria	Only high quality and policy compliant secrets (e.g. passwords) are accepted. Old secret must be invalid after successful secret change. Old secret must be kept and new one should not be valid when intercepting the secret change dialog. Account must be locked after a certain number of wrong login attempts. A correct username and secret can not be used to login to a locked account (also a wrong one can not be used). Locked account must be explicitely enabled by the user or is automatically enabled after a certain time
Environmental requirements	
Special Procedural Requirements	
Execution Procedure	Login(correct username, correct secret). Try to change secret to low quality one. Change secret to high quality one. Try to login with old secret after secret change. Start to change secret and intercept secret change dialog. Try to login with new secret from the intercepted secret change dialog. Perform several unsuccessful logins until account is locked. Try to login with correct username and secret. Reset locked account. Try to login with correct username and secret
Precondition	
Postcondition	

Description	This test assures that the mechanism works correct in case of wrong login attempts and weak secret changes
Purpose	Test the one factor authentication on an advanced level
Type	requirement

Table 5.4: Secret change

Attribute	Value
ATC_ID	ATC2
ATCName	Input boundary
Input	slightly lower/shorter than lower input boundary input. exact lower boundary input. slightly above/longer than lower boundary input. slighty lower/shorter than upper boundary input. exact upper boundary input. slightly above/longer than upper boundary input
Output	Mechanism behavior
Evaluation Criteria	Mechanism must work as specified during the whole test case (see evaluation criteria of other test cases for this mechanism)
Environmental requirements	
Special Procedural Requirements	
Execution Procedure	Test the mechanism with all input data and also its combinations when multiple inputs have to be given (6 powered X possibilities; X = number of input possibilities/fields)
Precondition	
Postcondition	
Description	This test assures that the mechanism works correct on its boundarys and slightly beyond/beneath them
Purpose	Test the mechanism if it works correctly on its boundaries and around them
Type	boundary

Table 5.5: Input boundary

Attribute	Value
ATC_ID	ATC7

ATCName	Authentication Robustness
Input	Correct Username. Wrong Username. Correct Secret. Wrong Secret. Special characters. Forbidden characters. None input. Extreme long input. Vendor Username and Secret (if exist)
Output	Login successful or forbidden
Evaluation Criteria	Login was only successful with correct username and correct secret
Environmental requirements	
Special Procedural Requirements	
Execution Procedure	Login(correct username, correct secret). Login(wrong username, correct secret). Login(correct username, wrong secret). Login(wrong username, wrong secret). Also try this with special characters, forbidden characters, long input and none input (4 powered X combinations; X = number of input possibilities/fields). Try the vendor username and secret. Mix all input data as combinations (10 powered X; X = number of input possibilities/fields)
Precondition	
Postcondition	
Description	This test assures that the authentication mechanism does not fail when used outside its specification
Purpose	Test the authentication outside its specification
Type	robustness

Table 5.6: Authentication Robustness

As it can be seen, 4 atomic test cases (table 5.3, 5.4, 5.5 and 5.6) are used. Each of these cases aims on a different area of the requirement. This is achieved due to the usage of diverse testing techniques. Subsequently we give the rational for each test case.

Generic one factor authentication test (table 5.3)
This test covers the primary functionality of the requirement, which is to restrict access to known and correct authenticated users. Even tough this test aims on the basic functionality, it also includes coverage for some generic security issues of such a requirement. These security relevant issues are for example the encryption of the transmission of the login data, or the response time of the system in case of successful or unsuccessful login attempts. This

test can be classified as *usage-based*, as it is mostly based on actions and inputs which are used during the usage for its dedicated purpose.

Secret change (table 5.4)

The change of secrets or especially passwords is important as it is a common and security relevant task. Due to that this atomic test case covers the functionality of the secret change mechanism. As it can be seen in table 5.4, it covers not only the normal usage of the mechanism. This test includes also more specific requirements of the mechanism, as for example the quality of the changed password or the atomic behavior of this function. As the shown test case is mainly influenced by the given requirements for a secret change mechanism, it can be declared as *requirement*-based.

Input boundary (table 5.5)

Boundary tests aim on the boundaries of a system and the areas around them. As the test cases have a black-box oriented view on the observed system, this test case has its focus on the boundaries of input data. Due to that this test case includes boundary values for the system which are used to measure the systems behavior when pushed to its limits. It can be seen that the given test case in table 5.5 is a generic description of how to conduct such boundary tests. This is on the one hand caused due to the need for reusing the test case. On the other hand the test case is still on a sophistication level which enables even non-security aware testers to map the test case to a certain situation. Hence the above described balance between sophistication level and applicability of the test case is kept. Regarding the given test case it can be classified as *boundary* focused.

Authentication Robustness (table 5.6)

When examining an authentication mechanism due to the above described atomic test cases, the behavior of the mechanism for not specified situations has not been tested yet. This is covered by the test case shown in table 5.6. This test case aims on an area which lies beyond the system specification. Hence it provides a hunch of the system behavior in this area. As it can be seen the test case refers explicitly to forbidden input data as it should state out how the system reacts to it. This is a typical *robustness* test, as earlier described in section 3.3.

When comparing the recently described atomic test cases to the functional testing techniques in section 3.3, it can be seen that nearly every testing technique is used. The *Equivalence partitioning* technique is the only missing one. This is caused due to the structure of the approach. The *Equivalence partitioning* technique further relies on other techniques which are used for actually conducting the tests. For example most of the presented atomic test cases include different partitions, such as *correct password* and *incorrect password*, or *allowed characters* and *forbidden characters*. However, these tests have a closer relationship to one of the other testing techniques. Hence

it can be said that *Equivalence partitioning* should be seen as a top-level technique, which relies on other techniques. Due to that it is inapplicable at this level for many requirement. All other techniques are used by the test cases. Still, this should never be seen as an assurance that the test cases cover every potential fault. The comparison to the testing techniques should rather be seen as an indication for the coverage, which gives a hunch not an assurance that the requirement is tested good enough by the developed test cases. This is further discussed in chapter 6.

5.2.1.3 Composition of Test Case Specifications

When the test case specification and the atomic test cases are created, it is necessary to create a linkage between them. As earlier described, the test case specification of a requirement is determined by a certain security level. In this case two test case specifications are defined as shown in table 5.2. The basic test case specification provides only a rough coverage of the requirements specification, while the advanced one aims on a higher sophistication level. Further they aim on different regression levels. The composition of atomic test cases according to their aggregating test case specifications is shown in table 5.7.

Test Case Specification	Atomic Test Case
Basic one factor authentication	Generic one factor authentication test (table 5.3)
Advanced one factor authentication	Secret change (table 5.4)
Advanced one factor authentication	Boundary test (table 5.5)
Advanced one factor authentication	Authentication robustness (table 5.6)

Table 5.7 Mapping between the test case specifications (table 5.2) and atomic test cases

As it can be seen, the basic test case specification consists of one atomic test case which tests for the generic functionality of the requirement, while the advanced test case specification contains three atomic test cases, which aim on different parts of the requirement. Hence the correlating regression-levels differ. The basic test case specification is on regression *Level 3*. This leads to a higher probability of conduction of this test case during regression tests compared to the advanced test case specification, as it is on regression *Level 2*. This reflects the different aims of the test case specifications. The basic one provides only an overview of the principal functionality of the requirement, while the advanced one aims on additional security relevant issues. The security level for both specifications can be stated as *Level 1*.

This classification may seems to be too high for the basic specification, but it has to be mentioned that both specifications have to be conducted together to achieve this security level. A seperated execution can only be performed when it comes to regression tests. The accomplishable security level is also limited due to the requirement itself. In this case the maximum security level of the given requirement is *Level 1*, as a higher security level would require a more sophisticated security mechanism. This maximal achievable security level is determined by the SMRT for each mechanism. Further it has to be mentioned that the security levels are only achieved on the confidentiality and integrity dimensions, as the requirement is neither in charge, nor has the possibility to provide protection on the availability dimensions. As above described, the requirement belongs to the *User Authentication* category and according to the SMRT a requirement in this category can provide protection only on the C and I dimensions.

Summarized it was shown how to create test case specifications and atomic test cases according to a given requirement. It can be seen that this is not a trivial task, which requires expertise about the requirement, related security issues and also the functional security testing approach itself. However, the structure of the security testing approach and the presented process from section 4.5 provide support for creating content. This enables a later usage also by non-security experts.

5.2.2 Example B - SSL/TLS

The requirement shown in table 5.8 is part of the *Data integrity transfer protection-* , as well as the *Data confidentiality transfer protection* category. This is caused due to the fact that the requirement does not only provide confidentiality, but also integrity of transfered data at the same time. As the functional security testing approach is created with an aim on flexibility, this fact can be represented in the database without ambiguity.

Name	Description
SSL/TLS	Enables encryption of data between systems

Table 5.8 ExampleB Requirement

The requirement is in comparison to the above presented requirement in subsection 5.2.1 more specific. It directly refers to a technical standard, which

can be found in [IET99]. Hence, also the test cases can be more focused than the above presented ones. However, it has to be mentioned that the tests should be easy to conduct also by non-security experts. The feasibility is a criteria, which should be taken into account during the whole process for the development of test cases. Another important point is the effort, which has to be applied, to conduct the tests. When having a strict technical specification, it is easy to create a huge amount of test cases which cover every detail. During the execution of this tests it comes down to the mentioned feasibility problem. As all available resources are limited, too detailed test cases would lead to a consumption of this resources by testing technical details instead of covering the top level functionality of requirements. Hence a balance between the level of detail and the necessary effort to conduct these tests must be kept.

5.2.2.1 Test Case Specifications

According to the above presented requirement in table 5.8, the following test case specifications can be identified:

Name	Description
SSL/TLS without CAC approved algorithm	Tests if the SSL implementation fulfills the technical specification
SSL/TLS with CAC approved algorithm	Tests if the SSL implementation fulfills the technical specification

Table 5.9 Test Case Specifications for table 5.8

According to [Tel07b], the requirement provides protection from *Level 2* to *Level 3* (dependent on the used cryptographic algorithm). Due to that the test case specifications are split up according to this difference in the achievable security levels. In comparison to the two test case specifications in subsection 5.2.1, the split is caused by different security- not regression levels. The actual achieved security- and regression levels of the specifications are later described during this subsection. According to the process in section 4.5, these levels have to be kept in mind at this point as they influence the later derivation of atomic test cases.

5.2.2.2 Atomic Test Cases

In regard to the specification ([IET99]) of the requirement presented in table 5.8, the following atomic test cases can be identified:

Attribute	Value
ATC_ID	ATC202
ATCName	SSL/TLS Transfer
Input	Date (e.g text)
Output	
Evaluation Criteria	Data transmission must be encrypted. If transfer is manipulated during transmission it must be recognized by receiver.
Environmental requirements	Server with SSL support. Client with SSL support. Network sniffer. Packet modifier (i.e. Proxy).
Special Procedural Requirements	Record transmission
Execution Procedure	Send data between client and server. Manipulate data during transmission.
Precondition	
Postcondition	
Description	This test assures that the SSL/TLS data transmission is encrypted and can not be manipulated
Purpose	Test SSL/TLS data encryption and integrity
Type	usage-based

Table 5.10: SSL/TLS Transfer

Attribute	Value
ATC_ID	ATC203
ATCName	SSL/TLS Closed Connection Reestablishment
Input	Session ID from closed SSL session, Session ID from old but still valid SSL session
Output	
Evaluation Criteria	Server must reject SSL connection establishment with closed session id or offer new connection (new session ID or none). Server must accept reestablishment of old but valid session with corresponding session id
Environmental requirements	Server with SSL support, Client with SSL support

Special Procedural Requirements	Record transmission
Execution Procedure	Client sends hello_request with old and closed session id to server (same chipher and compression method as in old session). Client sends hello_request with old but valid session id to server (same chipher and compression method as in old session).
Precondition	Session foreplay recorded
Postcondition	
Description	This test assures that the SSL/TLS Server does not accept the reestablishment of closed connections and can handle the reestablishment of old but valid connections
Purpose	Test SSL/TLS Server handling of closed/old connections
Type	usage-based

Table 5.11: SSL/TLS Closed Connection Reestablishment

Attribute	Value
ATC_ID	ATC204
ATCName	CAC Approved algorithm
Input	
Output	
Evaluation Criteria	The config file for the data transfer protection must only contain secure chipher options (CAC approved) and a minimum key-length of 128 bit (cipher independent). Only such chiphers are accepted for the session establishment
Environmental requirements	System with data transfer protection support
Special Procedural Requirements	
Execution Procedure	Open config file for data transfer protection implementation
Precondition	
Postcondition	
Description	This test assures that the data transfer protection is correctly configured
Purpose	Test data transfer protection config
Type	requirement

Table 5.12: CAC Approved algorithm

Attribute	Value
ATC_ID	ATC205
ATCName	SSL/TLS Client Handshake
Input	
Output	
Evaluation Criteria	Connection is closed on any errors (connection aborted, timeout, wrong order of message exchange) during handshake. Only strong chipers are offered by client. Minimum key-length of 128 bit. Server certificate must be valid and issued by a trusted certification authority. Invalid server certificates are not accepted. The DNS name of the server is compared to he DNS name on the certificate. A downgrade to a less secure chipher or SSL/TLS version is not accepted. SSL version 2 or less should not be used.
Environmental requirements	Client with SSL support, Server with SSL support
Special Procedural Requirements	Record transmission
Execution Procedure	Client starts SSL/TLS session establishment to server with valid certificate and invalid certificate (criteria see in EvaluationCriteria). Abort SSLT/TLS session establishment. Server sends packages during session establishment in wrong order (i.e ChangeChipherSpec before client has sent Finished package). Client start SSL/TLS session establishment to a server which only provides weak chiphers (less than 128 Bit strength) or older SSL/TLS version than the one used by the client.
Precondition	
Postcondition	
Description	This test assures that the SSL/TLS implementation in the client can handle security relevant handshake parameters properly
Purpose	Test SSL/TLS client implementation correctness
Type	robustness

Table 5.13: SSL/TLS Client Handshake

Attribute	Value
ATC_ID	ATC206
ATCName	SSL/TLS Server Handshake
Input	
Output	
Evaluation Criteria	Connection is closed on any errors (connection aborted, timeout, wrong order of message exchange) during handshake. Only strong chipers are offered by server. Minimum key-length of 128 bit. Server certificate must be valid and issued by a trusted certification authority. The DNS name of the server is equal to he DNS name on the certificate. A downgrade to a less secure chipher or SSL/TLS version is not accepted. SSL version 2 or less should not be used.
Environmental requirements	Server with SSL support, Client with SSL support
Special Procedural Requirements	Record transmission
Execution Procedure	Client starts SSL/TLS session. Clients aborts session during handshake. Client starts new SSL/TLS session and sends package in wrong order during session establishment (i.e Finished package before ChangeChipherSpec). Client with older SSL/TLS version than the one used by the server tries to etablish a session. Client tries to negotiate a weak chipher (less than 128 Bit strength) during SSL/TLS handshake.
Precondition	
Postcondition	
Description	This test assures that the SSL/TLS implementation in the server can handle security relevant handshake parameters properly
Purpose	Test SSL/TLS server implementation correctness
Type	robustness

Table 5.14: SSL/TLS Server Handshake

Attribute	Value
ATC_ID	ATC207
ATCName	SSL/TLS Package Boundaries

Input	Packages with message length slightly less, equal and slightly more than 2 powered 14 bytes (further called invalid packages)
Output	
Evaluation Criteria	Package is not accepted by any party. Connection may be aborted by receiving party. (both is permitted to pass the test)
Environmental requirements	Server with SSL support, Client with SSL support
Special Procedural Requirements	Record transmission
Execution Procedure	Client starts SSL/TLS session. Clients sends invalid packages to server. Client closes SSL/TLS session. Cliend starts SSL/TLS session. Server sends invalid packages to client.
Precondition	
Postcondition	
Description	This test assures that the SSL/TLS implementation operates correct on its boundaries
Purpose	Test SSL/TLS implementation behavior on its boundary
Type	boundary

Table 5.15: SSL/TLS Package Boundaries

The atomic test cases are now explained in detail. It has to be mentioned that the functional testing techniques, as well as known problems and security issues from [USE96] and [And01] have influenced the crafting of these cases. Hence they cover most of the significant spots of the requirements specification.

SSL/TLS Transfer (table 5.10)
This atomic test case aims on the main task of the requirement, to provide an encrypted communication over an untrusted network. As *SSL/TLS* does provide protection for data confidentiality as well as data integrity during the transfer over a network, this test covers both areas. Even tough the requirement can be tested in detail, such general tests of the main target of the requirement are indispensable for testing its correct functionality. As this case tests for the behavior of the implementation in a typical usage scenario it an be classified as *usage-based*.

SSL/TLS Closed Connection Reestablishment (table 5.11)
The reestablishment of old sessions is part of the *SSL/TLS* specification. This is especially important for security testing as an implementation failure in

this part can lead to serious security problems. Hence this functionality is covered by a test cases. However, it has to be mentioned that the test case can only aim on the correct behavior of this feature for some situations. An attacker with malicious intend could find situations in which the requirement does not behave as specified. Due to the focus of this test case it can be classified as *usage-based*. This is caused due to testing procedure, which is strongly influenced by the intended usage of a connection reestablishment.

SSL/TLS Approved algorithm (table 5.12)

As above mentioned, the *SSL/TLS* requirement can achieve different maximum security levels. This is caused due to specifications given by the SMRT, which prescribes the kind of used algorithms. Hence the used algorithm determines the exact security level. Due to that a test case is created to check for the used algorithms and their compliance with the regulations given by the SMRT. As it can be seen in table 5.12, the configuration file of the implementation is checked for the used algorithms which are further checked for compliance with the security level specifications. For security *Level 2* no such regulations have to be taken into account, while it is necessary to use only *Crypto Advisor Council (CAC)* [Tel], [TC07] approved algorithms to achieve a *Level 3* classification. As this reflects a requirement for the implementation, the test case can be classified as *requirement*-based.

SSL/TLS Client Handshake (table 5.13)

When observing the *SSL/TLS* specification in order to detect security issues, a critical point is the establishment of new connections. Due to that this test case aims on security problems, which might occur in this stage. As the server and the client have slightly different tasks in the establishment process, they are addressed by separated test cases. As it can be seen in table 5.13, the criteria for evaluating the success of the test case are quite long compared to other atomic test cases. This is caused by the level of detail of this test in order to detect implementation misbehavior. However, it would be possible to go deeper into the transfered packages and check each single one for its correctness and potential failures. As the specification describes every single piece of exchanged information this could be done without any obstacles. When doing so the above mentioned balance between the necessary effort and the level of detail of a test case has to be kept in mind. Hence the test case shown in table 5.13 does observe the implementation not on such a level of detail. According to the aim of the test on the robustness of the establishment process it can be classified as *robustness* test.

SSL/TLS Server Handshake (table 5.14)

In comparison to the above described test for the robustness of the connection establishment from a client point of view, this test has the same focus in regard to the implementation of the server. Due to that it has slightly similar content, but targets at a different role within the establishment pro-

cedure. The main focus of this test lies on the correct use of security relevant parameters and the handling of faults. Another important point is the acceptance of client connections with weak algorithms or old protocol versions. On the one hand a server should enable connections to a whole range of different clients, all having diverse *SSL/TLS* implementations and versions. On the other hand it should provide that the transfer is protected. Due to this complementary requirements compromises have to be maid. Instead of accepting every weak cryptographic algorithm or old and unsafe protocol version, a minimum level of both are required. This trade-off has to be made to provide protection on the given security level.

SSL/TLS Package Boundaries (table 5.15)
The last atomic test case aims on the boundaries of the *SSL/TLS* specification. A limit which is set in the standard described in [IET99], is the length of a message. According to that a classical boundary test focus on the behavior of the implementation on, and around this boundary. It would be possible to define more tests which focus on such boundaries, as for example most of the exchanged data during the handshake procedure have to have a fixed size or content. On the other hand this would generate more test cases, which would mostly not be executed due to limited resources during the testing process.

Summarized it can be seen that the presented atomic test cases cover some security relevant spots of the requirement. Although it would be possible to dive deeper into technical details, the showed level allows a time-efficient conduction of the tests while still taking important technical issues into account. In regard to the functional testing techniques from section 3.3, it can be stated out that every of the techniques, except the *Equivalence Partitioning* is used. The reason for that is the same as described in subsection 5.2.1. Some of the techniques are even covered by multiple test cases. In comparison to the above presented atomic test cases in subection 5.2.1, the recently described ones have a stronger technical focus and test for specific details of the implementation. As already mentioned at the beginning of this section, this is caused by the availability of a strict specification of the implemented requirement.

5.2.2.3 Composition of Test Case Specifications

As the atomic test cases and the test case specifications are already created, it is now possible to link them together. As mentioned in the above subsection, this includes not only a mapping between them, but also the determination of the security- and regression levels. The linkage between the test case specifi-

cations from table 5.9 and the recently presented atomic test cases is shown in table 5.16.

Test Case Specification	Atomic Test Case
SSL/TLS without CAC approved algorithm	SSL/TLS Transfer (table 5.10)
SSL/TLS without CAC approved algorithm	SSL/TLS Closed Connection Reestablishment (table 5.11)
SSL/TLS without CAC approved algorithm	SSL/TLS Client Handshake (table 5.13)
SSL/TLS without CAC approved algorithm	SSL/TLS Server Handshake (table 5.14)
SSL/TLS without CAC approved algorithm	SSL/TLS Package Boundaries (table 5.15)
SSL/TLS with CAC approved algorithm	SSL/TLS Transfer (table 5.10)
SSL/TLS with CAC approved algorithm	SSL/TLS Closed Connection Reestablishment (table 5.11)
SSL/TLS with CAC approved algorithm	SSL/TLS SSL/TLS Approved algorithm (table 5.12)
SSL/TLS with CAC approved algorithm	SSL/TLS Client Handshake (table 5.13)
SSL/TLS with CAC approved algorithm	SSL/TLS Server Handshake (table 5.14)
SSL/TLS with CAC approved algorithm	SSL/TLS Package Boundaries (table 5.15)

Table 5.16 Mapping between the test case specifications (table 5.9) and atomic test cases

It can be seen that most of the atomic test cases are used by both test case specifications. The aim of the first specification is to achieve a security *Level 2*, both on the confidentiality and the integrity dimension. Due to that high classification, the requirement is tested properly. The second specification aims on security *Level 3* on the same dimensions. Hence a more secure implementation is necessary, which implies also more comprehensive tests. According to the SMRT, the approval of the used cryptographic algorithms is the core requirement to achieve a *Level 3* classification. To meet this requirement, the second specification includes an additional test case which aims on this fact. According to the high security levels of both test case specifications, their regression level is set to *Level 2*. This prescribes a conduction of all test cases in most regression tests.

Subsumed it can be seen that there exists a difference when creating test case specifications and atomic test cases for generic requirements, as demonstrated in subsection 5.2.1, or for accurate specified ones as it has been demonstrated in this subsection. Both types of requirements have different aims which leads further to different test cases. Hence this need for a different outcome must be kept in mind when conducting the process for creating content.

Chapter 6
Validation of the Functional Security Testing Approach

When observing the functional security testing approach from chapter 4 and the generated test cases from chapter 5, a need for measuring both arises. To measure the quantitative aspect, which refers to the completeness of coverage of requirements by their test cases, a method which includes the functional security testing techniques can be used. The qualitative aspect can be measured according to the criteria, which were described in chapter 1. There it is stated that the developed approach should fulfill some parts of the *Common Criteria* [ISO07] as shown in figure 1.1. In addition to this deep and specific evaluation, other relevant security standard from chapter 2 are used for creating a more holistic validating of the qualitative aspects of the approach. The practical usage of the approach is another criteria, which is of special importance for *TeliaSonera*. Due to all of these measures not only the functional security testing approach itself, but also the developed content and its practical usage is validated.

This chapter starts with a validation of the approach against the mentioned parts of the common criteria and other important security standards from chapter 2. Subsequently an approach for measuring the coverage of requirements by test cases is presented. The last validation part gives insight in the use of the approach in a real project. Hence all these parts represent a comprehensive validation of the approach against its requirements, which were described in chapter 1.

6.1 Validation against the Common Criteria

In order to validate the developed approach against the families and classes of the *Common Criteria* (CC), which are relevant in this case (figure 1.1),

these parts have to be observed in more detail. In the tests class (*Class ATE: Tests*) of the CC it is stated out that:

> Testing provides assurance that the TSF [TOE Security Function] behaves as described (in the functional specification, TOE [Target of Evaluation] design, and implementation representation). [...] The emphasis in this class is on confirmation that the TSF operates according to its design descriptions. This class does not address penetration testing [...] [ISO07]

Hence it can be said that this class and its included families are relevant for assessing the functional security testing approach. The different families are:

- ATE_COV: Coverage
- ATE_DPT: Depth
- ATE_FUN: Functional tests
- ATE_IND: Independent testing

Further it is described in the CC that these families can be separated according to the persons or roles that are aimed on. The first three families (Coverage, Depth, Functional tests) target on the developer and are named *Developer testing*. The remaining family (Independent testing) aims on the evaluator and is called *Evaluator testing*. *TeliaSonera* has a developers point of view, which the functional security testing approach should support. Hence only the developer testing families are taken into account. Subsequently we discuss these families and the validation of the approach according to them. It has to be mentioned that the validation is concerned about the ability of the approach to provide support for systems to achieve an EAL2 classification. Hence the validation aims on the application of the approach for functional security testing, not on the components of the approach itself.

6.1.1 ATE_COV: Coverage

In the CC this family is described as concerning about the fulfillment of a systems specification. It is stated that:

> This family establishes that the TSF has been testes against its functional specification. [...] [ISO07]

As shown in figure 1.1 the level to achieve for the approach is *Level 1*. The requirements for this level are that some of the *TOE Security Function Interfaces (TSFI)* have been tested. Regarding the developed approach, this criteria can be seen as fulfilled, as the approach aims on the functional testing of security requirements according to their specifications. Even though the fundamental aim of this family is fulfilled, it can be observed in more detail.

In the CC it is further described that the following actions and artifacts must be performed and created [ISO07]:

Developer action elements The developer shall provide evidence of the test coverage.

Content and presentation elements The evidence of the test coverage shall show the correspondence between the tests in the test documentation and the TSFIs in the functional specification.

Evaluator action elements The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

The *Developer action elements* requirements are fulfilled as the coverage of the test cases is assessed in the *Analyze* activity of the approach's process (section 4.2). Further the coverage of the included security requirements by its test cases is validated in section 6.3. The *Content and presentation elements* refer to the mapping of requirements to its functional test cases. This represents a test for their specifications fulfillment. As stated out in chapter 4, this mapping is one of the core components of the approach and directly influences the *Design test plan* activity.

The *Evaluator action elements* refer to the kind of provided information for the evaluator and are in this case of no further concern, as *TeliaSonera* is the developer and not the evaluator. Summarized it can be said that the functional security testing approach meets every requirement to support systems to be classified as *Level 1* compliant to this family.

6.1.2 ATE_DPT: Depth

As shown in figure 1.1 it is not necessary to fulfill any component of this family to have an EAL2 compliant system. However, the developed approach provides the fulfillment of requirements of this family. Hence this part is examined now in more detail. This family aims on the level of detail the TSF is tested by the developer [ISO07]. In the CC it is described what has to be done to achieve a *Level 3* classification in this family:

> The subsystem and module descriptions of the TSF provide a high-level description of the internal workings, and a description of the interfaces of the modules, of the TSF. [ISO07]

The high-level description of the internal workings, as well as the description of the interfaces are given through the requirements and test cases in the approach. Hence this aim can be stated out as fulfilled. When drilling deeper into this family, the following sub-requirements are described [ISO07]:

Developer action elements The developer shall provide the analysis of the depth of testing.

Content and presentation elements The analysis of the depth of testing shall demonstrate the correspondence between the tests in the test documentation and the TSF subsystems and modules in the TOE design. The analysis of the depth of testing shall demonstrate that all TSF subsystems in the TOE design have been tested. The analysis of the depth of testing shall demonstrate that all TSF modules in the TOE design have been tested.

Evaluator action elements The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

The *Developer action elements* part refers to an analyzation of the tests according to their depth. This can be achieved by an analysis of the test cases by the developer and the *Analyze* activity from section 4.2. The *content and presentation elements* require to take aspects outside the approach into account. This depends on the system picture. Assumptions have to be made about the specification and the complete codification of the system by its requirements. Each codified security requirement can be tested by the approach. This can be said as the extendibility of the approach provides that missing test data for a requirement can be added. Another aspect is the protection level a requirement is tested on. As this level further defines the rigor a requirement is testes with, it also affects the depth of the tests which is determined by the business need as shown in section 4.1. Hence it is possible that requirements are completely tested for each system components they apply. The *Analyze* activity can further show the depth of the tests and can be used to assess the depth of testing against the specification. The *Evaluator action elements* are the same as in the previous family *ATE_COV: Coverage*, where they are already described.

Summarized it has been shown in this part, that all requirements to provide a *Level 3* classification for systems are met.

6.1.3 ATE_FUN: Functional tests

This family is about the correct documentation and conduction of tests. The level to achieve is according to figure 1.1 *Level 1*. When regarding the top level aim of this level it is written in the CC that:

> The objective is for the developer to demonstrate that the tests in the test documentation are performed and documented correctly. [ISO07]

As the documentation and conduction of tests is supported by the functional security testing approach, it can be said that the main target of this family is

already met. Subsequently a more detailed description of the requirements for this level is given [ISO07]:

Developer action elements The developer shall test the TSF and document the results. The developer shall provide test documentation.

Content and presentation elements The test documentation shall consist of test plans, expected test results and actual test results. The test plans shall identify the tests to be performed and describe the scenarios for performing each test. These scenarios shall include any ordering dependencies on the results of other tests. The expected test results shall show the anticipated outputs from a successful execution of the tests. The actual test results shall be consistent with the expected test results.

Evaluator action elements The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

The *Developer action elements*, as well as the *Content and presentation elements* are both supported by the developed approach. Documentation about the executed tests is provided by the *Test result* artifact from section 4.2 and further the statistical information table of the approach. The included test cases contain not only information about the expected test results and outputs. They also provide a mapping between requirements and test cases in regard to dependencies between them. Hence the pre-codified nature of the security requirements and their related test cases allows a simpler and more comprehensive documentation. The *Compare results to test cases* activity (section 4.2) provides fulfillment of the requirement for assuring the consistency of test results. The last point, *Evaluator action elements*, is equal to the other families.

Summarized it can be said that the developed functional security testing approach does not only achieve the required sophistication level, which we described in chapter 1. It also provides the fulfillment of the *ATE_DPT* family on *Level 3*, as shown in figure 6.1. Due to the above discussed evaluation of the approach and its support for the functional security testing of systems it can be said that the validation against the mentioned parts of the CC has been successful and the structure of the approach fulfills the given requirements. Further not only the requirements for the testing part of an EAL2 classification are fulfilled by applying the approach, as it also includes the additional *ATE_DPT* family. Hence it can be declared as EAL2+ compliant.

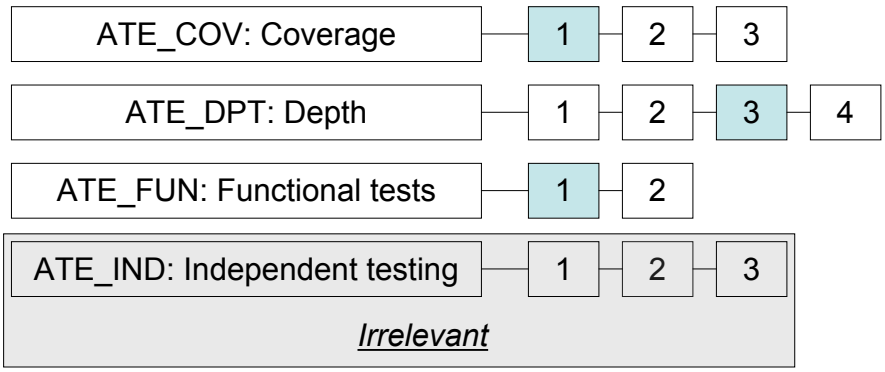

Fig. 6.1 Common Criteria, Class ATE: Tests, adapted from [ISO07]

6.2 Validation against other relevant parts of standards

As already mentioned in chapter 2, not only the CC has influenced the functional security testing approach, but other standards contributed as well. In this section we discuss the influence of these standards on the functional security testing approach and compare the compliance goals from chapter 2 with the actual approach.

ISO/IEC 27002 [ISO05c] and its *Asset classification and control* security control is addressed by the functional security testing approach due to the use of information classification and further the SMRT (see section 4.1). This further influences the *Test Plan* artifact of the approach. In section 2.1 we identified another relevant part for this work. *Systems development and maintenance* is concerned about mostly technical security topics which play an important role for the development and maintenance of systems. Hence this part has its major impact on the approach's content, the test cases, not on the design of the approach itself. Still, some parts of this security control have been used for the approach itself. *Security requirements of systems* and *Security requirements analysis and specification* are both concerned about assuring that the appropriate level of security from a business perspective is met. This objective is targeted by the functional security testing approach due to the use of *TeliaSonera's* information classification and its impact on the *Test Plan* artifact for specified requirements.

IEEE Std 829 [ISO98] has been used as a base for structuring and organizing test cases in the database of the functional security testing approach. The recommendations and guidelines provided by this standard are therefore integrated into a core part of the approach, which we further discussed in section 4.3.

ISO/IEC 21827 [Pro02] includes the *Process Area (PA) 11 - Verify and Validate Security*. As we already described in chapter 2, it targets on two main goals [Pro02]:

- Solutions meet security requirements
- Solutions meet the customers operational security needs

The first goal is addressed by the approach's capability of defining a linkage between security requirements and test cases to test for the correct functionality of the specified requirements. Further the *Test Plan* artifact captures the security requirements of a system and the *Analyze* activity of the approach's process aims on evaluating if the security requirements are actually met. The second issue of the mentioned process area is taken care of by the information classification and the SMRT. The information classification derives the technical protection needs due to the business needs and further determines the strength and kind of mechanism from the SMRT. This assures that the specified security needs of a system are aligned with the operational security needs of an organizations business.

6.2.1 Validation against ITIL

As we discussed in section 2.1, some parts of the ITIL standard [ITI08] are important for this thesis and have influenced the development of the functional security testing approach. Subsequently, we show how the developed approach fulfills the set compliance goals from section 2.1.

Service Warranty [CHR+07] is a concept from the strategic part of ITIL and is fulfilled by the approach due to the use of information classification and the SMRT which further influences the *Test Plan* artifact (see section 4.2. This assures the technical fitness of a system for its later usage from a business perspective.

In the design stage of the ITIL life-cycle we identified the proactive activities of *Availability Management* as compliance goals for this work. This issue is addressed by the proactive nature of the approach which provides to improve the overall quality of systems it is applied to before they are put into operation. This quality improvement is achieved due to the *Test Plan* artifact for proactive planning and design and the *Analyze* activity for recommendations concerning the security (including availability) of systems from a business perspective given due to the information classification and the *Test Plan*. Another part in this ITIL life-cycle stage is *Information Security Management (ISM)*. ISM aims on confidentiality, integrity and availability (C,I,A) issues. It requires the proper management of these aspects

to achieve an adequate level of C,I and A according to the organization's business needs. This is addressed by the functional security testing approach from the very beginning due to the use of *TeliaSonera's* information classification, which captures the business needs for protecting its systems.

The remaining relevant ITIL part for this work is situated in the service transition stage. The *Service Validation and Testing* process is concerned about appropriate testing and validation of IT solutions according to an organization's business risks [CHR+07]. Thereby the critical issue is to define the *appropriate* level of testing and validation. To solve this issue the functional security testing approach relies on the given information classification which captures business risks and transforms them into protection needs. These protection needs are further used by the approach's *Design test plan* activity and *Test Plan* artifact to determine the appropriate level of testing.

In this section we discussed the influence of some security standards on the development of the functional security testing approach. Further we validated the relevant parts, which we already described in chapter 2, against these standards and best practices. Summarized we showed that the functional security testing approach fulfills the compliance goals from section 2.1.

6.3 Validation of requirements coverage

To assess the coverage of requirements by its test cases, different approaches can be used. The opportunities reach from a simple counting system till expert based observations. As the counting of test cases has no meaning to assess the coverage and the expert based observation can be declared as too cost and time intensive, a meaningful but also cost oriented way of assessing must be found. As already stated in chapter 5, the functional testing techniques (section 3.3) can be taken into account to create evaluation criteria. Hence these techniques are used for this assessment as well. When observing security requirements and their correlating test cases, the kind of used testing techniques by the test cases must be kept in mind. This can be seen as an indication of the completeness of coverage of a requirement by its test cases. If a requirement does not use a certain functional security testing technique for its test cases, there must be a reason why this technique is not suitable for this requirement. Otherwise some test cases may be missing. Due to that the completeness of coverage of a requirement by its linked test cases can be evaluated.

It has to be mentioned that this approach to assess the completeness is heuristic. It can only give a hunch of the grade of coverage and should not be seen as assurance that the test cases check for the entire functionality of a requirement. A list of requirements and their use of functional test techniques by their atomic test cases is shown in table 6.1. This table gives an overview of some requirements. As the functional security testing approach is designed to be continuously extended, it should either be seen as a basis to build-up than a complete listing. An important point to mention is that *Equivalence-partitioning* is the only technique which can not be assessed directly due to observing the atomic test cases. As already mentioned in section 5.2, this technique cannot be used on the atomic test case level. Hence the usage of this technique is assessed manually in table 6.1, while all other techniques are directly extracted from the database.

Requirement	Techniques				
	Usage-based	Requirement-based	Boundary	Robustness	Equivalence-partitioning
RQ0:Standard one factor authentication	X	X	X	X	X
RQ1:Strong one factor authentication	X	X	X	X	X
RQ100:Basic discretionary Access Control	X				X
RQ1000:Basic Public-Private-Key based Key management		X		X	
RQ1001:Basic Shared Private Key based Key management		X		X	
RQ1002:Advanced Public-Private-Key based Key management		X		X	
RQ1003:Advanced Shared Private Key based Key management		X		X	
RQ1004:Extensive Public-Private-Key based Key management		X		X	
RQ1005:Extensive Shared Private Key based Key management		X		X	
RQ101:Advanced discretionary Access Control		X			X
RQ102:Basic Role-based Access Control	X			X	X
RQ103:Advanced Role-based Access Control	X	X		X	X
RQ104:Mandatory Access Control	X	X			X

RQ2:Two factor authentication	X	X	X	X	X
RQ200:Non-cryptopgraphic based Checksum		X			
RQ201:MultiProtocol Label Switching		X		X	
RQ202:SSL/TLS	X	X	X	X	
RQ203:IPsec	X	X		X	
RQ3:Physical two factor authentication	X	X	X	X	X
RQ400:Weak cryptographic integrity protection	X				
RQ401:Write once read many memory		X		X	
RQ402:Smartcard	X				
RQ403:Strong cryptographic integrity protection	X	X			
RQ500:Basic stored file encyrption	X	X		X	
RQ501:Strong encryption of selected areas or files	X	X		X	
RQ502:Strong encryption of entire disk	X	X		X	
RQ503:Smartcards as storage device	X	X		X	
RQ600:Slow Incident Detection	X	X		X	
RQ601:Continous Incident Detection	X	X		X	
RQ602:Real-time Incident Detection	X	X		X	
RQ603:Active response Intrusion Detection System with host generated data	X	X		X	
RQ604:Active response Intrusion Detection System with host and network generated data	X	X		X	
RQ605:Constantly human supervised Intrusion Detection system	X	X		X	
RQ700:ISO-9000		X			
RQ701:ISO/IEC 1799/2005		X			
RQ702:IRT-planning		X			
RQ703:CERT-planning		X			
RQ800:Basic traceability		X			
RQ801:Advanced traceability		X			
RQ802:Extensive tracability		X			
RQ803:Legal complying tracability		X			
RQ900:Local Backups	X	X			
RQ901:Cold standby systems	X	X		X	
RQ902:Physically seperated Backups	X	X			
RQ903:Warm standby systems	X	X		X	
RQ904:Location seperated Backups	X	X			

RQ905:Hot standby systems	X	X		X	

Table 6.1: Requirement coverage by functional test techniques

It can be seen in table 6.1 that not all requirements are fully covered by every test technique. This can be caused due to the following reasons.

Inapplicability of the testing technique In some cases a testing technique can be useless or inapplicable for a certain requirement. Such a relation is mostly caused due to the nature of the requirement or the testing technique. For example it is not possibly to apply a boundary or robustness test to the requirement for *ISO-9000* compliance. These test techniques are simply inadequate and thus inapplicable for this kind of requirement. There is no reasonable case in which such a technique can be used to test this requirement. As shown in table 6.1, such a situation is given quite often especially when it comes down to generic requirements as described in section 5.2.

Ambiguity of test case classification As defined by the database schema described in chapter 4, an atomic test case can be classified according to the functional testing techniques from section 3.3. Thereby the test case must be classified by a best-fit matching as one of the given techniques. In some cases this can lead to ambiguity as an atomic test case probably uses more than one technique, and just one of them can be picked as classification for the whole test case in regard to the best-fit matching. While this could be avoided by a split of the test cases, this may not be done due to feasibility reasons. For example it can be practicable to include some usage aspects in a requirements test, as they sometimes can aim on similar behavior. Even tough a solution for this problem would be to allow an atomic test case to be classified as using multiple testing techniques, this is not supported by the approach. This design decision is caused by the fundamental assumption that an *atomic test case should be atomic*. Hence it should just aim on testing a certain small part of a requirement. Due to that one testing technique should be enough to classify an atomic test case. Another advantage of this restriction is that the developer of new content is forced to keep the atomic test cases focused and reusable. This reusability further leads to a higher benefit from the approach in the long-run. However, this ambiguity is prevented in the currently included atomic test cases and was just mentioned as additional explanation in case of a further content extension and re-evaluation of the quantitative aspect of the approach.

Summarized it can be said that there exist some reasons why a requirement is not covered by all testing techniques. However, the aim of the developed test cases is to create a holistic but superficial coverage of the given requirements. As it can be seen in table 6.1, such a coverage is given for the included requirements. Hence it can be stated out that the developed content base provides satisfactory coverage for the given requirements.

6.4 Validation in a real-life project

To test the practical applicability of the functional security testing approach, a project has been chosen. As the aim of this validation is to get first feedback, an *Uppdrag* (engl.: Assignment) is considered as the most convenient way for performing an initial reality check. This term describes a small project in *TeliaSonera*. *Min Familj* (engl.: My family) is such an *Uppdrag*, which takes place during the creation of this thesis and hence has been chosen for applying the functional security testing approach. Subsequently some facts and figures about the project are presented:

- Mobile telecommunication package for families
- Includes a shared calendar which can be used by all family members over the internet and mobile devices
- Maximal budget of 2000.000 SEK (Swedish Crowns)
- Mobility project
- Conducted in the organizational line

The critical part in this project is the calendar application. Hence the functional security testing approach is used for it. An information classification, which was followed by a gap analysis (see section 4.1), performed by an internal security expert showed a need for protecting the personal data entered in the calendar at level 2-2-3 (C,I,A). The data entered into this system can be seen as important from a privacy perspective. The protection of them is critical to avoid bad reputation for *TeliaSonera* and provide customer satisfaction. Hence such a high classification can be justified. Subsequently the steps of the testing process from chapter 4 are described. Thereby the focus is not on the process itself, as this was already described in the mentioned chapter. In order to create a validation of the approach we discuss the conduction of the process steps in the *Min Familj* project.

6.4.1 Design test plan

During the creation of the test plan, not only the functional security testing approach has been considered. In addition also security checklists (see section 4.2) have been included in the test plan. The use of the approach for identifying test cases according to given requirements and security levels turned out to be complicated in the beginning. A critical issue, which was mentioned by the creator of the test plan, was the missing graphical interface. Due to that it was necessary to perform SQL statements to extract the needed data from the approach. A less skilled tester would probably not have been able to do this task properly. As this critical issue was considered by me and my superviser *Albin Zuccato* in advance, such a graphical interface is currently under development and already reached a maturity stage which solves the mentioned problem.

When the tester extracted the relevant data from the database he identified a number of 8 TestCaseSpecifications in total for the given requirements. 6 of these specifications matched to the given requirements without any objections. The remaining 2 were valuable as well, but had some shortcomings, as some aspects of the requirements were not considered in detail. However, due to the use of the test case specifications the tester mentioned the positive effect of discovering missing or ambiguous parts in the system's specification. According to [Boe81] this reduces the need for change in later stages of the project. Thus costs can be lowered and the time for conducting projects can be reduced.

Due to the missing graphical support, *3 hours* have been spent by the tester for understanding the approach and executing necessary SQL requests. For a first draft of the test plan *2 hours* were needed. Finally the tester finished the document after *1 hour* of verification in the end. According to the experience of the tester, it takes an average of 6 to 8 hours to create a proper test plan.

The 3 hours in the beginning can be considered as start activity, which is obsolete when using the graphical user interface in the future. Hence the average time for the test plan creation can be halved from more than 6 to 3 hours by using the functional security testing approach. This is a significant improvement for *TeliaSonera*.

6.4.2 Run program with test data

As the project included only requirements which were already in the content base of the approach, there was no need for designing new test cases. Hence

the activities *Design test cases* and *Prepare test data* had no importance in this case. This possibility to skip the activities is described in section 4.2.

The execution of the atomic test cases was done without any major problems. An issue which was mentioned by the executing tester was the level of the atomic test cases. It was proposed that it would be beneficial to make the atomic test cases shorter and hence ”more atomic”. This would increase the possibilities for reusing them and reassemble more detailed test case specifications. This issue was already considered earlier during the creation of the approach. As a design decision which was mainly driven by constraints due to limited resources, the level of the atomic test cases is not as granular, as it could be. However, this is part of the further development of the approach for *TeliaSonera*.

6.4.3 Compare results to test cases

As the test cases include a description of the expected results, this task was conducted without any problems. A field of difficulty for generic requirements could be the interpretation of the criteria for the evaluation of the tests, but in this case the tester did not mention it as a problem. Hence the comparison of the results was an easy task in this case which did not put too much demands on the tester's capabilities.

6.4.4 Analysis

The analysis of the results lacked of graphical support. In this project this was not brought up by the tester as a problem, which is mainly caused due to the small number of test cases and therefore few test results to analyze. Still, for a later usage in larger projects this is likely to be a critical issue. Hence the interface has to be extended to sufficiently support also this last activity.

Summarized it can be said that the use of the approach did not only facilitate the whole testing process for the tester, but also improved the quality level of the specification while lowering the costs and the time of execution. Hence the functional security testing approach fulfills its task to improve *TeliaSonera's* testing process. Therefore the validation in a real project can be stated as successful.

Chapter 7
Conclusion

The aim of this work was to show that *it is possible to create a correlation between functional security tests and given security requirements, which enables a direct mapping of test cases to their underlying requirements*. In course of this thesis we discussed how an approach can be designed to solve this problem. Thereby the aim was not only on developing such an approach in general, but integrating it into the given structures of *TeliaSonera*. Due to that not only the research question was answered, also the benefit from using the developed artifacts could be significantly increased. The developed approach helps *TeliaSonera* to gain assurance about the fulfillment of their security requirements in a more cost-efficient and effective way as done before.

In a company of the size of *TeliaSonera*, constantly a large number of products and systems are in development. Due to that it is impossible for the security experts of the company to individually support each project with security expertise. Hence the approach is designed to encapsulate security knowledge, which makes functional security testing suitable to be conducted by non-security experts. This leads to another cost and time advantage for *TeliaSonera*, as the spare and expensive security experts can focus their knowledge in the approach and hence spread it throughout the whole organization with reduced effort.

Although the fundamentals and core of this thesis are academic, its outcome has also practical value. The developed approach did not only undergo theoretical validations. Its was further used and validated in a real-world project. Hence it was possible to evaluate if the theoretical assumptions sustain the transfer to operational reality. As shown in this thesis the approach fulfills the demands put on it and its usage leads to the above mentioned benefits as well in reality.

7.1 Contributions of this work

The aim of this thesis is to show that it is possible to create a mapping between requirements and test cases. Even though this has been the main target during the working process, it is not the only contribution. During the process of creating this work some additional findings have been made. It can be said that those are all focused on the security testing topic, but they are still situated in different areas. These areas reach from theoretical ones, like the *Definition of Functional Security Testing*, to practical ones like the *Atomic Test Cases*. Some of them were developed due to the necessity to have a foundation for the functional security testing approach, while others are a simple outcome of the development of the approach itself. Hence the presented contributions are settled along the whole creation process of this work, which is shown in figure 1.2. Due to the diverse areas the contributions are situated in, they are of different importance for *TeliaSonera* and the scientific community. While the theoretical findings are likely to be of higher value for scientific research, the more practical oriented ones are considered more valuable for *TeliaSonera*. However, this work includes both types and hence provides useful information for its whole target audience. Subsequently the contributions of this work are described.

Definition of Functional Security Testing In chapter 3 an overview of different security testing approaches is given. As the focus of this work is on functional security testing this chapter especially targetes on that. Due to the missing definition of functional security testing, one is presented in section 3.3.

Automatizing parts of the testing process The developed functional security testing approach from chapter 4 includes not only static aspects like the database schema, but also aims on supporting the dynamic testing process as described in section 4.2. This idea of how to support and automatize the process is another contribution of this work.

Atomic Test Cases The quantitative part of this thesis is mainly represented by the developed content for the functional security testing approach, especially by the atomic test cases. The atomic test cases are the accomplishable tests which aim to test a certain area of one or more requirements. The developed tests have a superficial but holistic coverage of the included requirements and can be seen as contribution of this thesis.

Link between Requirements and Test Cases The main aim of this thesis is to show that *it is possible to create a correlation between functional security tests and given security requirements, which enables a direct mapping of test cases to their underlying requirements*. The creation of this linkage is a contribution of this work. The link between the business domain and

the needed protection mechanism for security is mentioned in section 4.1 and given by the SMRT. This thesis shows how to create the link between the requirements and their test cases in order to provide proper functional security tests for them.

7.2 Outlook and further research

The current development stage of the presented functional security testing approach allows a use as repository for functional test cases. It can further be used to support a testing process through automation. Additional to the current functionality some ideas of how to extend the approach in future came up. Extension should in this case be seen as extension of the qualitative component of the approach itself, rather than the development of more test cases for the current structures. These further research ideas are subsequently described.

7.2.1 Attack Patterns

Attack Patterns, as described in [MIT08a], show a schema of how to organize common attacks to reduce the complexity of execution due to standardization. These attacks do not reach a sophistication level as a penetration test does. In comparison to functional tests, which do not include tests with malicious intend, they have a stronger *attackers* point of view.

Figure 7.1 shows this relation. As it can be seen, the attack patterns lie in the fuzzy area between the functional- and penetration tests. This is caused due to the fact that a strong delineation between these areas can not be drawn as the intermediate parts include various elements of both testing techniques. However, projects like [MIT08a] provide a standardized structure for attack patterns. Hence it should be possible to integrate them into the functional security testing approach what would lead to an extension which can cover most testing needs of *TeliaSonera*. As already mentioned in section 4.2, systems sometimes need to be tested more comprehensive than it could be achieved by functional tests alone. Due to that an extension of the approach with this kind of simple and *canned* penetration attacks would provide higher benefit for *TeliaSonera*.

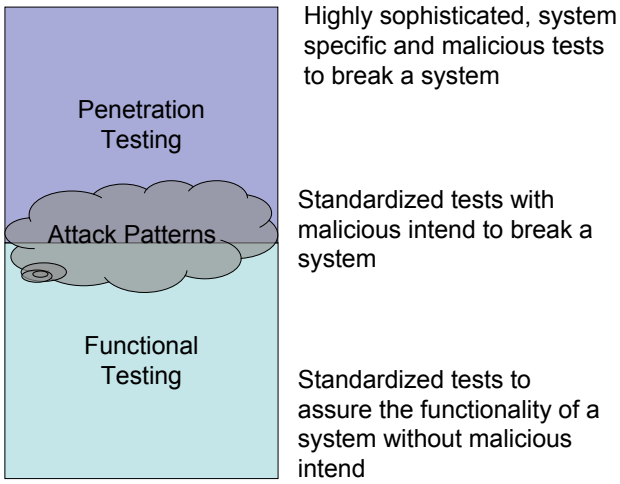

Fig. 7.1 The relation between functional- , attack pattern based- and penetration testing, adapted from [TT07]

7.2.2 Software Weakness Collection

The *Common Weakness Enumeration (CWE)* [MIT08b] provides a collection of known software weaknesses. It was already mentioned in section 4.4 as idea for extending the top level schema. Another project which has the same aim as the CWE can be found in [OWA08]. An integrated link to a known weakness would allow various new opportunities to assess and control the completeness of a requirements coverage by its test cases. Such a validation can take the coverage of a requirement according to known weaknesses into account. Due to that a coverage of them by the contained test cases in the database would be possible. Hence it can be said that the further integration of the CWE or another collection of software weaknesses would provide completely new possibilities to validate the quantitative quality of the testing approach. This leads to a higher benefit for *TeliaSonera*. Hence it should be considered for further research and development in this area.

7.2.3 More Automatization

The described process in section 4.2 shows how a standard testing process, as mentioned in section 2.5, can be supported by the functional security testing approach. This support is mostly given due to automatization of ac-

tivities and the providence of artifacts. The opportunity to execute the testing process faster and more cost efficient is the core value for *TeliaSonera*. Due to that it should be considered as further research area. To achieve a higher grade of automatization different ideas came up during the creation of this work. A standardized derivation from an identified protection need to a requirement would provide a higher automatization in the first steps of the process, thus enabling a direct mapping from a protection need over a requirement till the tests which have to be conducted. In the later activities the execution of tests is one of the major time consuming activities. To achieve a higher level of automatization in this activity a methodology for creating machine understandable and executable test cases should be introduced. As *TeliaSonera* deals mostly with heterogeneous systems and hence diverse requirements, such an automation can in all likelihood not be accomplished for all test cases. However, specific requirements which are based on strict technical specifications can be tested by such automated tests as already partially done within *TeliaSonera* [Tel07a] or demonstrated in [AS08].

Summarized it can be said that a higher grade of automation can be directly transformed into value created due to cost- and time reductions.

7.2.4 Compliance to standards

As already mentioned in chapter 2, some security standards are used for the functional security testing approach. The approach enables a shift of *TeliaSonera's* product life cycle towards better compliance to [ISO05c], [Pro02], [ITI08] and [ISO07]. As already some aspects of these standards are included into *TeliaSonera's* practices and the developed approach aims on a sound integration into the structures of *TeliaSonera*, some parts of the standards have indirectly influenced its development (e.g. section 4.1). Still, this level of compliance could be improved in a future enhancement of the functional security testing approach. Hence we see the further advancement of the approach for improving its compliance to the mentioned standards as an important point.

As overall conclusion of this thesis it can be said that an approach for functional security testing has been developed to improve the testing- and therefore the whole product life cycle of *TeliaSonera*. Thereby it is important to mention that this is not only a theoretical construct as it has been validated in practice with the outcome that it works and directly creates value for the company.

References

[And01] Ross Anderson. *Security Engineering - A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., 2001.

[AS08] A-SIT. Zentrum fuer sichere informationstechnologie austria ssl-check clients/server, 2008.

[BDG+98] David Banks, William Dashiell, Leonard Gallagher, Charles Hagwood, Raghu Kacker, and Lynne Rosenthal. Software testing by statistical methods. Technical report, National Institute of Standards and Technology - Information Technology Laboratory, 1998.

[Bes] Best Management Practice for Project, Programme Risk and Service Management. *ITIL V3 Glossary - Glossary of Terms and Definitions*.

[Bis03] Matt Bishop. *Computer Security - Art and Science*. Addison-Wesley, 2003.

[Boe81] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[BS05a] British Standards Institute. *BS 7799-1 Information technology. Security techniques. Code of practice for information security management*, 2005.

[BS05b] British Standards Institute. *BS 7799-2 Information technology. Security techniques. Information security management systems. Requirements*, 2005.

[BS06] British Standards Institute. *BS 7799-3 Information security management systems - Guidlines for information security risk management*, 2006.

[BSI] Bundesamt für sicherheit in der informationstechnologie - http://www.bsi.de.

[CHR+07] Alison Cartlidge, Ashey Hanna, Colin Rudd, Ivor Macfarlane, John Windebank, and Stuart Rance. *An Introductory Overview of ITIL V3*. itSMF - The IT Service Management Forum and Best Management Practice for Project, Programme Risk and Service Management, 2007.

[CJ06] R.D. Craig and S.P Jaskiel. *Systematic Software Testing*. Northwood MA: Artech House Publishers, 2006.

[CW07] Brian Chess and Jacob West. *Secure Programming with Static Analysis*. Pearson Education Inc., 2007.

[Dij70] Edsger W. Dijkstra. Structured programming. In *Software Engineering Techniques*. NATO Science Committee, August 1970.

[DRP99] Elfriede Dustin, Jeff Rashka, and John Paul. *Automated Software Testing*. Addison-Wesley, 1999.

[EU] European-Union. Education, audivisual & culture executive agency - erasmus mundus.

[FSI] Fortify-Software-Inc. Fortify 360, http://www.fortify.com. 2008.

[GHB04] It-grundschutzhandbuch. Technical report, Bundesamt für Sicherheit in der Informationstechnologie, 2004.

[Gol06] Dieter Gollmann. *Computer Security*. John Wiley & Sons Ltd., 2006.

[HL02] Michael Howard and David LeBlanc. *Writing Secure Code*. Microsoft Press, 2002.

[HM04] Greg Huglund and Gary McGraw. *Exploiting Software - How to break code*. Addison-Wesley, 2004.

[IET98] IETF. Security architecture for the internet protocol. Technical report, Internet Engineering Taskforce - Network Working Group, 1998.

[IET99] IETF. The tls protocol. Technical report, Internet Engineering Taskforce - Network Working Group, 1999.

[ISO98] International Standards Organization. *IEEE Std 829-1998 - IEEE Standard for Software Test Documentation*, 1998.

[ISO05a] International Standards Organization. *ISO/IEC 17799:2005 Information technology - Security techniques - Code of practice for information security management*, 2005.

[ISO05b] International Standards Organization. *ISO/IEC 27001:2005 Information technology - Security techniques - Information security management systems - Requirements*, 2005.

[ISO05c] International Standards Organization. *ISO/IEC 27002:2005 Information technology - Security techniques - Code of practice for information security management*, 2005.

[ISO07] International Standards Organization. *ISO/IEC 15408 Information Technology - Security Techniques - Evaluation Criteria for IT Security - Common Criteria*, 2007.

[ITI08] Official it infrastructure library website - http://www.itil-officialsite.com, 2008.

[Jan05] Girish Janardhanudu. White box testing. Technical report, U.S. Department of Homeland Security and Cigital Inc., 2005.

[JC83] Martin J. and McClure C. *Software Maintenance: The problem and Its Solutions*. Prentice Hall, 1983.

[KFN99] Cem Kaner, Jack Falk, and Hung Quoc Nguyen. *Testing Computer Software*. John Wiley & Sons Inc., 1999.

[MA07] MySQL-AB. Mysql database 5.045 http://www.mysql.com, 2007.

[McG06] Gary McGraw. *Software Security - Building Security in*. Addison-Wesley, 2006.

[Mea06] Nancy R. Mead. Security requirements engineering. Technical report, U.S. Department of Homeland Security and Carnegie Mellon University, 2006.

[MIT08a] MITRE. Common attack pattern enumeration and classification. Technical report, MITRE Corporation and U.S. Department of Homeland Security, http://capec.mitre.org, 2008.

[MIT08b] MITRE. Common weakness enumeration. Technical report, MITRE Corporation and U.S. Department of Homeland Security, http://cwe.mitre.org/, 2008.

[MR05] C. C. Michael and Will Radosevich. Risk-based and functional security testing. Technical report, U.S. Department of Homeland Security and Cigital Inc., 2005.

[NIS] National institute of standards and technology - http://www.nist.gov.

[OMG07] Open Management Group OMG. Unified modeling language. Technical report, 2007.

[OWA08] OWASP. Open web application security project http://www.owasp.org, 2008.

[PCCW93] Mark C. Paulk, Bill Curtis, Mary Beth Crissis, and Charles V. Weber. Capability maturity model-se, version 1.1. Technical report, Carnegie Mellon Software Institute, 1993.

[Per91] William E. Perry. *Quality Assurance for Information Systems*. John Wiley & Sons, Inc., 1991.

[PM04] Bruce Potter and Gary McGraw. Software security testing. *IEEE Security & Privacy*, 2004.

[PP03] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing*. Prentice Hall, 2003.

[Pro02] Systems Security Engineering Capability Maturity Model (SSE-CMM) Project. *ISO/IEC 21827 Systems Security Engineering - Capability Maturity Model*. International Standards Organization, 2002.

[Som04] Ian Sommerville. *Software Engineering*, volume 7. Addison-Wesley, 2004.

[SSE] Systems security engineering - capability maturity model - http://www.sse-cmm.org.

[SU] Stockholm-University. Department of computer and systems sciences.

[TC07] TeliaSonera and Per Christoffersson. List of evaluation reports. Internal Techreport, 2007.

[Tea06] CMMI Product Team. Cmmi for development, version 1.2 - improving processes for better products. Technical report, Carnegie Mellon Software Engineering Institute, 2006.

[Tel] TeliaSonera. Crypto advisor council.

[Tel02] TeliaSonera. Test case and attack pattern definition per system/network an mechanism. Internal Techreport, 2002.

[Tel07a] TeliaSonera. Checkmate. Internal Tool, 2007.

[Tel07b] TeliaSonera. Security mechanism reference table. Internal Document, 2007.

[Tel07c] TeliaSonera. Security test plan. Internal Document, 2007.

[Tel07d] TeliaSonera. Test coverage assesment. Internal Document, 2007.

[Tel08] TeliaSonera. http://www.teliasonera.com, 2008.

[Tho03] Herbert H. Thompson. Why security testing is hard. *IEEE Security & Privacy*, 2003.

[TT07] Babak Tighnavard and Björn Torstensson. A software security testing framework - introducing the attack pattern matrix. Master's thesis, Royal Institute of Technology - Sweden, 2007.

[USE96] USENIX Association. *Analysis of the SSL 3.0 Protocol*, volume 2 of *USENIX Workshop on Electronic Commerce*, November 1996.

[VM02] John Viega and Gary McGraw. *Building Secure Software*. Addison-Wesley, 2002.

[vW07] Kenneth R. van Wyk. Adapting peneration testing for software development purposes. Technical report, Department of Homeland Security and Carnegie Mellon University, 2007.

[WT03] James A. Whittaker and Herbert H. Thompson. *How to break Security*. Addison-Wesley Longman, 2003.

[Zuc04] Albin Zuccato. Holistic security requirement engineering for electronic commerce. *Computer & Security*, 23, 2004.

[Zuc05] Albin Zuccato. *Holistic Information Security Management Framework for electronic commerce*. PhD thesis, Karlstads Universitet, 2005.

List of Figures

1.1 Common Criteria, Class ATE: Tests, adapted from [ISO07] . 3
1.2 Master Thesis Methodology 6

2.1 The ITIL service life-cycle, adapted from [CHR+07] 13
2.2 The Software Development Life Cycle (SDLC), adapted from [Som04] 17
2.3 The Secure Development Life Cycle (SDL), adapted from [McG06] 20
2.4 The aim of Testing, adapted from [Tho03] 23
2.5 Categorization of testing approaches 24
2.6 The Software Testing process, adapted from [Som04] 27

3.1 The aim of penetration Testing, adapted from [Tho03] 32
3.2 The aim of functional Testing, adapted from [Tho03] 34
3.3 The aim of regression Testing, adapted from [Tho03] 38

4.1 Functional Security Testing Approach Process 43
4.2 Extract of the testing standard from [ISO98] 46
4.3 Top level database schema for the functional security testing approach 50
4.4 Relational schema for the functional security testing approach 52
4.5 Process for the creation of content for the functional security testing approach 60

6.1 Common Criteria, Class ATE: Tests, adapted from [ISO07] . 90

7.1 The relation between functional- , attack pattern based- and penetration testing, adapted from [TT07] 104

List of Tables

4.1 Security Mechanism Reference Table (SMRT) [Tel07b] 41

5.1 ExampleA Requirement 67
5.2 Test Case Specifications for table 5.1 68
5.3 Generic one factor authentication test 70
5.4 Secret change 71
5.5 Input boundary 71
5.6 Authentication Robustness 72
5.7 Mapping between the test case specifications (table 5.2) and atomic test cases 74
5.8 ExampleB Requirement 75
5.9 Test Case Specifications for table 5.8 76
5.10 SSL/TLS Transfer 77
5.11 SSL/TLS Closed Connection Reestablishment 78
5.12 CAC Approved algorithm 79
5.13 SSL/TLS Client Handshake 79
5.14 SSL/TLS Server Handshake 80
5.15 SSL/TLS Package Boundaries 81
5.16 Mapping between the test case specifications (table 5.9) and atomic test cases 84

6.1 Requirement coverage by functional test techniques 95

Appendix A
Functional Security Testing Approach Database Schema

Listing A.1 MySQL Database Schema

```
1  DROP DATABASE `SecTestCases`;
2  CREATE DATABASE `SecTestCases`;
3  -- ------------------------------ SCHEMA
4  USE `SecTestCases`;
5  CREATE TABLE SecurityLevel (
6          SecLevel_ID      VARCHAR(255) PRIMARY KEY,
7          SecLevel         INTEGER NOT NULL,
8          SecLevelName     VARCHAR(255) NOT NULL,
9          SecLevelDescr    TEXT,
10         SecLevelType     ENUM("C","I","A") NOT NULL
11 )ENGINE=INNODB;
12
13 CREATE TABLE RegressionLevel (
14         RegLevel_ID      VARCHAR(255) PRIMARY KEY,
15         RegLevel         INTEGER NOT NULL,
16         RegLevelName     VARCHAR(255) NOT NULL,
17         RegLevelDescr    TEXT
18 )ENGINE=INNODB;
19
20 CREATE TABLE Mechanism (
21         Mech_ID          VARCHAR(255) PRIMARY KEY,
22         MechName         VARCHAR(255) NOT NULL,
23         MechDescr        TEXT
24 )ENGINE=INNODB;
25
26 CREATE TABLE Category (
27         Cat_ID           VARCHAR(255) PRIMARY KEY,
```

```
28          CatName             VARCHAR(255) NOT NULL,
29          CatDescr            TEXT
30  )ENGINE=INNODB;
31
32  CREATE TABLE Requirement (
33          Req_ID              VARCHAR(255) PRIMARY KEY,
34          ReqName             VARCHAR(255) NOT NULL,
35          ReqDescr            TEXT
36   )ENGINE=INNODB;
37
38  CREATE TABLE TestCaseSpecification (
39          TCS_ID              VARCHAR(255) PRIMARY KEY,
40          TCSName             VARCHAR(255) NOT NULL,
41          TCSDescr                    TEXT,
42          Precondition        TEXT,
43          Postcondition       TEXT
44  )ENGINE=INNODB;
45
46  CREATE TABLE TCSDependency (
47          TCS_ID              VARCHAR(255) NOT NULL,
48          TCSAnt_ID           VARCHAR(255) NOT NULL,
49          PRIMARY KEY(TCS_ID, TCSAnt_ID),
50          CONSTRAINT TCSDep_TCS_dependency FOREIGN KEY (TCS_ID)
51                  REFERENCES TestCaseSpecification (TCS_ID)
52                  ON UPDATE CASCADE
53                  ON DELETE CASCADE,
54          CONSTRAINT TCSDep_TCS_dependency2 FOREIGN KEY (TCSAnt_ID)
55                  REFERENCES TestCaseSpecification (TCS_ID)
56                  ON UPDATE CASCADE
57                  ON DELETE CASCADE
58  )ENGINE=INNODB;
59
60  CREATE TABLE AtomicTestCase (
61          ATC_ID                      VARCHAR(255) PRIMARY KEY,
62          ATCName                     VARCHAR(255) NOT NULL,
63          Input                       TEXT,
64          Output                      TEXT,
65          EvalCriteria                TEXT NOT NULL,
66          Environment                 TEXT,
67          SpecProcReq                 TEXT,
68          ExecProcedure               TEXT NOT NULL,
69          Precondition                TEXT,
```

```
70          Postcondition               TEXT,
71          Descr                       TEXT NOT NULL,
72          Purpose                     TEXT NOT NULL,
73          ATCType                     ENUM("boundary","requirement",
74                  "robustness","usage-based","equivalence") NOT NULL
75  )ENGINE=INNODB;
76
77  CREATE TABLE ATCDependency (
78          ATC_ID          VARCHAR(255) NOT NULL,
79          ATCAnt_ID       VARCHAR(255) NOT NULL,
80          PRIMARY KEY(ATC_ID, ATCAnt_ID),
81          CONSTRAINT ATCDep_ATC_dependency FOREIGN KEY (ATC_ID)
82                  REFERENCES AtomicTestCase (ATC_ID)
83                  ON UPDATE CASCADE
84                  ON DELETE CASCADE,
85          CONSTRAINT ATCDep_ATC_dependency2 FOREIGN KEY (ATCAnt_ID)
86                  REFERENCES AtomicTestCase (ATC_ID)
87                  ON UPDATE CASCADE
88                  ON DELETE CASCADE
89  )ENGINE=INNODB;
90
91  CREATE TABLE Cat2Mech (
92          Cat_ID  VARCHAR(255) NOT NULL,
93          Mech_ID VARCHAR(255) NOT NULL,
94          PRIMARY KEY (Cat_ID, Mech_ID),
95          CONSTRAINT Cat2Mech_Cat_dependency FOREIGN KEY (Cat_ID)
96                  REFERENCES Category (Cat_ID)
97                  ON UPDATE CASCADE
98                  ON DELETE CASCADE,
99          CONSTRAINT Cat2Mech_Mech_dependency FOREIGN KEY (Mech_ID)
100                 REFERENCES Mechanism (Mech_ID)
101                 ON UPDATE CASCADE
102                 ON DELETE CASCADE
103 )ENGINE=INNODB;
104
105 CREATE TABLE Mech2Req (
106         Mech_ID         VARCHAR(255) NOT NULL,
107         Req_ID          VARCHAR(255) NOT NULL,
108         PRIMARY KEY (Mech_ID, Req_ID),
109         CONSTRAINT Mech2Req_Cat_dependency FOREIGN KEY (Mech_ID)
110                 REFERENCES Mechanism (Mech_ID)
111                 ON UPDATE CASCADE
```

```
112                 ON DELETE CASCADE,
113         CONSTRAINT Mech2Req_Req_dependency FOREIGN KEY (Req_ID)
114                 REFERENCES Requirement (Req_ID)
115                 ON UPDATE CASCADE
116                 ON DELETE CASCADE
117 )ENGINE=INNODB;
118
119 CREATE TABLE Req2TCS (
120         Req_ID          VARCHAR(255) NOT NULL,
121         TCS_ID          VARCHAR(255) NOT NULL,
122         C               VARCHAR(255) NOT NULL,
123         I               VARCHAR(255) NOT NULL,
124         A               VARCHAR(255) NOT NULL,
125         RegLevel_ID     VARCHAR(255) NOT NULL,
126         PRIMARY KEY(Req_ID, TCS_ID, C, I, A, RegLevel_ID),
127         CONSTRAINT Req2TCS_Req_dependency FOREIGN KEY (Req_ID)
128                 REFERENCES Requirement (Req_ID)
129                 ON UPDATE CASCADE
130                 ON DELETE CASCADE,
131         CONSTRAINT Req2TCS_TCS_dependency FOREIGN KEY (TCS_ID)
132                 REFERENCES TestCaseSpecification (TCS_ID)
133                 ON UPDATE CASCADE
134                 ON DELETE CASCADE,
135         CONSTRAINT Req2TCS_Reg_dependency FOREIGN KEY (RegLevel_ID)
136                 REFERENCES RegressionLevel (RegLevel_ID)
137                 ON UPDATE CASCADE
138                 ON DELETE CASCADE,
139         CONSTRAINT Req2TCS_C_dependency FOREIGN KEY (C)
140                 REFERENCES SecurityLevel (SecLevel_ID)
141                 ON UPDATE CASCADE
142                 ON DELETE CASCADE,
143         CONSTRAINT Req2TCS_I_dependency FOREIGN KEY (I)
144                 REFERENCES SecurityLevel (SecLevel_ID)
145                 ON UPDATE CASCADE
146                 ON DELETE CASCADE,
147         CONSTRAINT Req2TCS_A_dependency FOREIGN KEY (A)
148                 REFERENCES SecurityLevel (SecLevel_ID)
149                 ON UPDATE CASCADE
150                 ON DELETE CASCADE
151 )ENGINE=INNODB;
152
153 CREATE TABLE TCS2ATC (
```

```
154         TCS_ID              VARCHAR(255) NOT NULL,
155         ATC_ID              VARCHAR(255) NOT NULL,
156         PRIMARY KEY (TCS_ID, ATC_ID),
157         CONSTRAINT TCS2ATC_TCS_dependency FOREIGN KEY (TCS_ID)
158                 REFERENCES TestCaseSpecification (TCS_ID)
159                 ON UPDATE CASCADE
160                 ON DELETE CASCADE,
161         CONSTRAINT TCS2ATC_ATC_dependency FOREIGN KEY (ATC_ID)
162                 REFERENCES AtomicTestCase (ATC_ID)
163                 ON UPDATE CASCADE
164                 ON DELETE CASCADE
165 )ENGINE=INNODB;
166
167 CREATE TABLE ConductedTests (
168         CT_ID                       VARCHAR(255) PRIMARY KEY,
169         ATC_ID                      VARCHAR(255),
170         TCS_ID                      VARCHAR(255),
171         Req_ID                      VARCHAR(255),
172         C                           VARCHAR(255),
173         I                           VARCHAR(255),
174         A                           VARCHAR(255),
175         RegLevel_ID                 VARCHAR(255),
176         Success                     BOOLEAN NOT NULL,
177         FailureReason               TEXT,
178         Descr                       TEXT,
179         Time                        TIME,
180         Costs                       INTEGER,
181         CONSTRAINT ConductedTests_Req2TCS FOREIGN KEY (TCS_ID, Req_ID,
182         C, I, A, RegLevel_ID)
183                 REFERENCES Req2TCS (TCS_ID, Req_ID, C, I, A,
184                 RegLevel_ID)
185                 ON UPDATE CASCADE
186                 ON DELETE CASCADE,
187         CONSTRAINT ConductedTests_ATC FOREIGN KEY (ATC_ID)
188                 REFERENCES AtomicTestCase (ATC_ID)
189                 ON UPDATE CASCADE
190                 ON DELETE CASCADE
191 )ENGINE=INNODB;
```

Appendix B
Functional Security Test Cases

B.1 CA0 - Authentication

Everything what authenticates belongs to this category

B.1.1 ME0 - One factor

Secret (e.g. password) is stored or transmitted in clear text. Default vendor passwords must never be used. Should not be used for new systems, only described for legacy systems. No matching requirement for ME0

B.1.2 ME1 - One factor

Secret (e.g. password) is NEITHER stored NOR transmitted in clear text. Default vendor passwords must never be used. Quality of passwords or other factors must be assured, e.g by using a policy that describes quality requirements. After a pre-determined number of logon attempts a user account the account is locked. A locked account requires the user to take action to enable an unlocked account or the account is unlocked after a time period (delay mechanisms).

B.1.2.1 RQ0 - Standard one factor authentication

Password, pin, etc.

- **TCS0 - Basic one factor authentication**: Tests if the one factor (e.g password) authentication works properly on a very basic level

  ATC0 - Generic one factor authentication test This test assures that the authentication mechanism works correct on a basic level and only allows correct authorized users to access the system

- **TCS1 - Advanced one factor authentication**: Tests if the one factor (e.g password) authentication works properly on an advanced level

  ATC1 - Secret change This test assures that the mechanism works correct in case of wrong login attempts and weak secret changes
  ATC2 - Boundary test This test assures that the mechanism works correct on its boundarys and slightly beyond/beneath them
  ATC7 - Authentication Robustness This test assures that the authentication mechanism does not fail when used outside its specification

B.1.3 ME2 - One factor strong authentication

One factor authentication where the secret is not repeated, i.e a new secret is generated for each session. For example a cryptographic challenge response mechanism (one time passwprd-OTP) or certificate, without PIN.

B.1.3.1 RQ1 - Strong one factor authentication

One time password, challenge response, certificates, etc.

- **TCS3 - Strong one factor authentication**: Test for the correct implementation of one-time, cryprographic and challenge-response authentication

  ATC1 - Secret change This test assures that the mechanism works correct in case of wrong login attempts and weak secret changes
  ATC2 - Boundary test This test assures that the mechanism works correct on its boundarys and slightly beyond/beneath them
  ATC3 - Authentication Logging This test assures that authentication activities are logged

  ATC0 - Generic one factor authentication test This test assures that the authentication mechanism works correct on a basic level and only allows correct authorized users to access the system

ATC4 - Challenge response, certificate and one time password This test assures that challenge response and one time passwords can not be reused and Certificates must be valid

ATC7 - Authentication Robustness This test assures that the authentication mechanism does not fail when used outside its specification

B.1.4 ME3 - Two factor strong authentication without a seperate physical token

For example a cryptographic challenge response mechanism or PC based certificates, protected by PIN.

B.1.4.1 RQ2 - Two factor authentication

Combination of one time password, challenge response, certificates, pin, password etc.

- **TCS4 - Two factor authentication**: Test for the correct implementation of one-time, cryprographic and challenge-response authentication

  ATC5 - Two factor authentication Test checks if a two factor authentication works correct

  ATC1 - Secret change This test assures that the mechanism works correct in case of wrong login attempts and weak secret changes

  ATC2 - Boundary test This test assures that the mechanism works correct on its boundarys and slightly beyond/beneath them

  ATC3 - Authentication Logging This test assures that authentication activities are logged

  ATC4 - Challenge response, certificate and one time password This test assures that challenge response and one time passwords can not be reused and Certificates must be valid

  ATC7 - Authentication Robustness This test assures that the authentication mechanism does not fail when used outside its specification

B.1.5 ME4 - Two factor strong authentication with seperate physical token

At least one of the factors must be a seperated physical token. For example using a physial OTP password generator (with PIN protection) which generates one time passwords.

B.1.5.1 RQ3 - Physical two factor authentication

Combination of one time password, challenge response, certificates, pin, password etc. One factor must be physical (e.g token)

- **TCS5 - Physical two factor authentication**: Test for the correct implementation of one-time, cryprographic and challenge-response authentication, where one of the factors must be a physical token

  ATC2 - Boundary test This test assures that the mechanism works correct on its boundarys and slightly beyond/beneath them

  ATC3 - Authentication Logging This test assures that authentication activities are logged

  ATC0 - Generic one factor authentication test This test assures that the authentication mechanism works correct on a basic level and only allows correct authorized users to access the system

  ATC6 - Physical two factor authentication Test checks if a physical two factor authentication works correct

  ATC5 - Two factor authentication Test checks if a two factor authentication works correct

  ATC7 - Authentication Robustness This test assures that the authentication mechanism does not fail when used outside its specification

B.2 Further categories

Due to security concerns just one example category is shown directly in this work. For an overview of the content of all categories please contact:

Albin Zuccato, PhD.
TeliaSonera Sweden
SE-123 86 Farsta, Sweden

Zusammenfassung

Funktionale Sicherheits- Tests können verwendet werden um sicherzustellen, dass die spezifizierte Sicherheitsfunktionalität korrekt funktioniert. Dabei liegt der Fokus nicht auf einem bösartigen Angriff auf das System, sondern auf der Erkenntnis, dass bestimmten Funktionen und Mechanismen vertraut werden kann. In dieser Arbeit präsentieren wir einen Ansatz für die Organisation und strukturierte Durchführung funktionaler Sicherheits- Tests. Dieser Ansatz ist flexibel gestaltet, und kann weiters für verschiedene und heterogene Systeme eingesetzt werden. Zusätzlich ist der in dieser Arbeit präsentierte Ansatz erweiterbar, so dass er an die spezifischen Bedürfnisse der Organisation und ihre Systeme angepasst werden kann. Um die grundlegende Praktikabilität der theoretischen Konzepte zu demonstrieren, präsentieren wir rudimentäre quantitative Daten für das Konzept. Diese erste Überprüfung ist notwendig, um die theoretischen Konzepte in der Praxis greifbar zu machen. Der Ansatz enthält eine Struktur, die Richtlinien und Vorgaben für die Erweiterung der Inhalte durch die Entwicklung von quantitativen Testdaten bietet. Dies ist eine Notwendigkeit auf Grund der Sensibilität dieser Aufgabe und der angenommenen Häufigkeit dieser Tätigkeit durch die Anpassung des Ansatzes an die spezifischen Bedürfnisse der einsetzenden Organisation.

Zur Bewertung der Validität und der praktischen Anwendbarkeit des Ansatzes wird eine Validierung durchgeführt. Diese ist in drei Stufen aufgebaut. Am Anfang zeigen wir eine Validierung der theoretischen Konzepte in Bezug auf die Konformität zu den Common Criterias [ISO07] und die Einhaltung anderer einschlägiger Normen und bewährter Praktiken. Anschließend demonstrieren wir die Vollständigkeit der Tests durch eine Überprüfung der Abdeckung der quantitativen Testdaten im Verhältnis zu den überprüfenden Sicherheitsfunktionen und Mechanismen. Als letzten Teil der Validierung, zeigen wir die Anwendbarkeit des Ansatzes in einem realen Projekt.

Zusammengefasst kann gesagt werden, dass diese Arbeit eine ganzheitliche Betrachtung von funktionalen Sicherheits- Tests bietet und einen flexiblen und erweiterbaren Ansatz für die Durchführung und Organisation solcher präsentiert.

Curriculum Vitae

Name:	Clemens Kögler
Geburtsdatum, - ort:	14.08.1986 in Wien

Ausbildung:

1992 - 1996	Volksschule GTS Köhlergasse
1996 - 2004	AHS Rosasgasse
2004 – 2005	Präsenzdienst beim Österr. Bundesheer
2005 – 2007	Bakkalaureatsstudium der Wirtschaftsinformatik an der Technischen Universität Wien und der Universität Wien mit dem Schwerpunkt „Electronic Commerce“
seit 07/2007	Masterstudium der Wirtschaftsinformatik an der Universität Wien
08/2007 – 05/2008	Auslandsaufenthalt an der University of Stockholm

Abschlüsse:

06/2004	Reifeprüfung mit Auszeichnung
06/2007	Bakkalaureat der Wirtschaftsinformatik mit Auszeichnung (Bakk. rer. soc. oec.)

Akademischer Werdegang:

04/2008 – 07/2008	Praktikum in einer Forschungsgruppe im Bereich Security & Mobility Services der TeliaSonera AB, Schweden
02/2009	Publikation mit „Dr. Albin Zuccato“ über „Functional Security Testing“ im Rahmen des „International Symposium on Engineering Secure Software and Systems“, Löwen, Belgien