Supporting Evolving Security Models for an Agile Security Evaluation

Wolfgang Raschke*, Massimiliano Zilli*, Philip Baumgartner†
Johannes Loinig†
Christian Steger*, and Christian Kreiner*

*Institute for Technical Informatics, Graz University of Technology, Graz, Austria
{wolfgang.raschke, massimiliano.zilli, steger, christian.kreiner}@tugraz.com
†NXP Semiconductors Austria GmhH, Gratkorn, Austria
{philip.baumgartner, johannes.loinig}@nxp.com

***Abstract*—At present, security-related engineering usually requires a big up-front design (BUFD) regarding security requirements and security design. In addition to the BUFD, at the end of the development, a security evaluation process can take up to several months. In today's volatile markets customers want to influence the software design during the development process. Agile processes have proven to support these demands. Nevertheless, there is a clash with traditional security design and evaluation processes. In this paper, we propose an agile security evaluation method for the Common Criteria standard. This method is complemented by an implementation of a change detection analysis for model-based security requirements. This system facilitates the agile security evaluation process to a high degree.**

I. Introduction

Traditional security engineering requires a big up-front design (BUFD) which includes the following engineering tasks: threat analysis, security requirements elicitation, security design and (security) architecture. Reviews ensure the consistency of these artifacts. In highly secure systems, the consistency of the security requirements and the security architecture is formally verified. Altogether, this constitutes a huge effort, before the implementation is even started upon. After the security design, the system is implemented according to requirements and design. Thereafter, evidence for security has to be provided in the form of documentation. This documentation is then delivered to the evaluation facility. The evaluation may take several months. If the feedback from the evaluator is negative, the system has to be re-designed. If this is the case, then a large amount of time and effort would be required. Moreover, the pressure to deliver the product to market is high. Summarizing, the challenges for security engineering are: early validation and time-to-market.

The software industry came up with the trend of agile development practices (see [2]), such as XP, Scrum, and Test-driven development. Agile methods focus on customer interaction and incremental software development. Feedback loops, such as customer on-site, pair programming and refactoring tend to minimize errors. Generally, agile processes react flexibly to changing customer requirements. In contrast, traditional processes tend to fulfill contractually specified requirements and are inflexible, by nature. Agile processes have demonstrated a high success rate in several software projects [3].

Unfortunately, agile methods are not well studied for high-level security engineering and evaluation. Case studies and success stories are lacking, especially for the Common Criteria standard [1]. Despite the fact that traditional security engineering seems to counter agile practices (see [4]), we think that a synthesis of both is not contradictory, if well considered.

Our contribution is the analysis of two processes which are possibly suitable for an agile security evaluation. Both processes are compared and one of them is then proposed for an agile security evaluation. In an agile evaluation process, all security properties of the system are kept in a model which is capable of creating documentation for an evaluation round. In the agile evaluation process only the increment of the model has to be reviewed. This has several benefits: the entire system does not have to be reviewed in each iteration. The evaluator starts with a small system and gains experience with the maturing of the system. Early feedback enables an early and thus more inexpensive correction of the system. With elaborated algorithms we are able to create a difference model for all changes during an increment. Additionally, we demonstrate the feasibility of automation by customizing existing open source software.

II. Background

A. Requirements for the Security-Relevant Agile Development Process

We developed this method mainly in order to gain improvements regarding time-to-market and early validation. In addition we strive to improve the semi-automation of the document creation with the model-based approach.

1) Time-to-Market: is one of the key success factors in the high-security business. In a traditional certification, the evaluation is accomplished after the implementation is finished. The time-to-market can significantly be reduced, if the evaluation parallels the development.

978-1-4799-6339-3/14/$31.00 © 2014 IEEE

ESPRE 2014, Karlskrona, Sweden

2) Early Validation: In the traditional certification approach, the evaluation starts late, when the product is already finished. If there is a negative evaluation result, the refactoring is expensive and seriously delays the completion of the product.

3) Semi Automation: The agile way of working states that communication shall be over documentation. Frequent interaction with the customer and building and adapting quickly a running system is imperative. Regarding the mentioned issues of a quick and flexible development, a documentation effort is counterproductive. In principle, the documentation could start right after the implementation: one would not have to care about documentation during the agile development but the benefit of time-to-market is lost. In this case the documentation and certification may last several months. In a highly competitive business this is a considerable delay. Moreover, also the advantage of early validation is lost.
It can be seen that the described security-relevant agile development process is a trade-off. However, much of the documentation effort can be automated: the model can be synchronized with the source code, the tests, and other relevant artifacts. A considerable degree of the documentation can then be created by a code-generator from the model.

B. Change Detection Analysis

A Change Detection Analysis (CDA) computes all changes between two versions. The CDA is more commonly known under the name *diff* analyis: diff[1] is a software that compares two versions of a text and marks the differences between them. It is a standard tool for software versioning and configuration management. The CDA for models is similar but does not make a line-based comparison of the text. Such a line-based comparison between two models is not sufficient because it does not take into account the structure of a model. In contrast, the CDA recognizes the type of the changed artifact. For example, the CDA recognizes, if and interface has changed in the source code. In this case, more related artifacts need to be reviewed: the regarding design, the tests and the like. If a change is not part of an interface, much less artifacts need to be taken into account. Textual diff tools do not have these capabilities. Basically, the CDA detects three different kinds of changes [5]:

- **Add:** Add a model element. A model element is a node in a model. It may have parent and child nodes.
- **Delete:** Delete an element.
- **Update:** Change of an element property. A property (attribute) belongs to an element.

C. Traceability Impact Analysis

The Traceability Impact Analysis (TIA) is an extension of the CDA. It takes all changed elements and finds all elements which are possibly affected by them. For this purpose, the TIA has to follow all dependencies (traces) of the changed elements. Thus, an element can be directly affected (by a changed element) or indirectly via several intermediate dependencies.

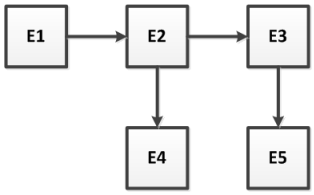

Fig. 1. Sample dependency graph. E4 is impacted directly by E2 and intermediatly by E1 (via E2).

1) Dependency Matrix: In order to perform a TIA, the dependency matrix has to be constructed. In (1) the matrix *D* represents the dependency graph shown in Fig. 1. In the matrix, each line lists the depending elements of a certain element. So, line 1 states that element E2 depends on element E1.

$$D = \begin{array}{c} \text{Element} \\ E1 \\ E2 \\ E3 \\ E4 \\ E5 \end{array} \begin{array}{c} \begin{array}{ccccc} E1 & E2 & E3 & E4 & E5 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \tag{1}$$

2) Reachability Matrix: The reachability matrix (2) can be computed from the dependency matrix. It states all direct and indirect dependencies between two elements. The reachability matrix can be used to calculate all impacts of a changed element.

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{2}$$

For example, in (3), the reachability matrix is multiplied with a vector that indicates that element E2 has changed (the *1* in the second row). The outcome of the multiplication is the vector *IS* that lists all impacted elements. So, if element E2 changes, element E3, E4 and E5 are impacted.

$$IS = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

[1] http://www.gnu.org/software/diffutils/

D. Delta Evaluation

The Common Criteria information statement on *Reuse of Evaluation Results and Evidence* [6] requires for a reuse of evaluation results the following evidence:

- Product and supporting documentation
- New Security Target
- Original Security Target
- Original Evaluation Technical Report
- Original Common Criteria Certificate
- Original Evaluation Work Packages

The document states:

> "The evaluation facility would be required to perform a delta analysis between the new security target and the original security target to determine the impact of changes on the analysis and evidence from the original evaluations."

and:

> "However, the evaluation facility conducting the current evaluation should not have to repeat analysis previously conducted where requirements have not changed nor been impacted by changes in other requirements."

Summarizing, this approach states the necessary documentation and the need for a Change Impact Analysis. A more detailed process for this evaluation is not given and has to be defined together with the evaluation facility.

E. Security Model

The security model describes the properties and relations of the developer evidence. Basically, the developer evidence contains the *Security Target* (ST), the design documentation, the implementation representation and tests. Such a model for a Common Criteria evaluation is very complex due to the very hierarchical composition of the Common Criteria artifacts. Therefore, we describe only the basic properties of such a model which are necessary for understanding the proposed evaluation approach (see Fig. 2).

The Security Target is the security claim and describes the operating context and how the TOE establishes its security. The ST is published and thus contains no detailed description of the design and implementation. The model of the ST contains the following main parts:

- *The Security Problem Definition* describes the environment of the TOE: threats, assumptions about its operation and its security relevant assets.
- *The Security Objectives* are high-level security goals which are then refined by the Security Functional Requirements.
- *The Security Functional Requirements* are a set of well understood requirements in a security domain and have to be fulfilled by the TOE.

The remaining developer evidence basically contains the design documentation, the implementation representation and tests. The evaluation has to prove that the TOE fulfills the security claim of the ST. For this reason, mainly the following developer evidence is necessary:

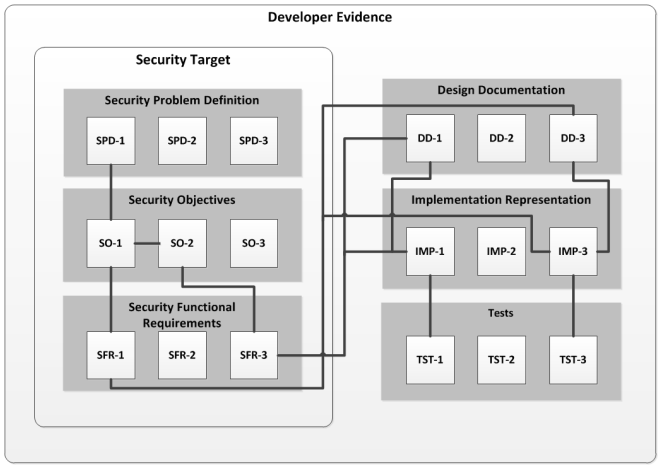

Fig. 2. Artifacts of the security model and traces between these artifacts.

- *The Design Documentation* contains the functional specification and the design specification.
- *The Implementation Representation* is the source code of the implementation.
- *Tests:* This part contains of the source code of the tests and also a corresponding test documentation.

III. Process Design

In this section we will explain and compare two possible processes for a security evaluation. The first one is more formal and suits a delta certification approach. The second is more informal and more appropriate for an agile security evaluation. What is the major difference between a delta and an agile evaluation process? The main difference is the frequency of the evaluation: the delta evaluation is based on a previous evaluation result which has been ascertained some time ago. Basically two product versions are compared. In the agile approach the evaluator becomes update documentation in each agile iteration (which lasts several weeks). Nevertheless, the certificate is only ascertained once, at the end of the product development lifecycle. The following processes are based on the change impact analysis process described by Bohner [7].

A. Process with Traceability Impact Analysis

In the following, we will describe the TIA process with an example. The resulting sets are listed in Table I. The graph in Fig. 1 shows the structure of the two versions N and N+1. The structure of the model does not alter during this increment. For the example, we assume, that the content of the node E1 has changed. The CDA detects all model differences between both versions. The output is the Starting Impact Set (SIA). In this case the SIA is the changed element E1: SIA = $\{E1\}$.

The TIA resolves then all traces and dependencies of the SIA and computes the so-called Candidate Impact Set (CIS). The CIS is the set of all possibly impacted artifacts. In this

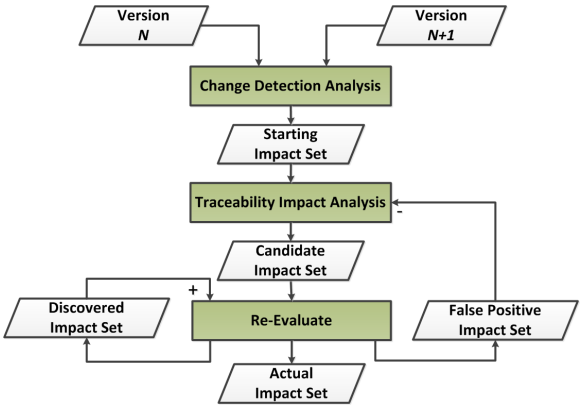

Fig. 3. Re-evaluation process with Traceability Impact Analysis. The Actual Impact Set is finally the re-evaluated set of software artifacts.

case, the CIS is: CIS = {E1,E2,E3,E4,E5}. Nevertheless, the CIS has to be manually re-evaluated:
(a) If the candidate has a possible impact, the corresponding evaluation artifacts need to be updated. The set of all re-evaluated artifacts form the Actual Impact Set (AIS). The Actual Impact Set in this example is: AIS ={E1,E2}.
(b) During the re-evaluation, new dependencies may be found: the Discovered Impact Set. Also the newly discovered impacts (DIS) have to be re-evaluated, again. In this example, the DIS is empty.
(c) If the candidate has no impact, it is part of the False Positive Impact Set (FPIS). In this case, a further security evaluation of the corresponding artifacts can be omitted. The FPIS in this example is: FPIS = {E3,E4,E5}.
Basically, the TIA predicts the AIS. This prediction is of course partly wrong, because new impacts are discovered (DIS) and some predicted impacts actually have no impact (FPIS). In (4) the relationships between the mentioned sets are stated:

$$AIS = CIS + DIS - FPIS \tag{4}$$

TABLE I
SAMPLE SETS FOR THE PROCESS WITH TRACEABILITY IMPACT ANALYSIS. THE CORRESPONDING EXAMPLE IS DESCRIBED IN SECTION III-A.

	Set	**Output Of**
Starting Impact Set	{E1}	CDA
Candidate Impact Set	{E1,E2,E3,E4,E5}	TIA
Discovered Impact Set	{}	re-evaluation
False Positive Impact Set	{E3,E4,E5}	re-evaluation
Actual Impact Set	{E1,E2}	re-evaluation

B. Process with Experiential Impact Analysis

We will describe the EIA process with an example. The resulting sets are listed in Table II. The graph in Fig. 1 shows the structure of the two versions N and N+1. As shown in Fig. 4, the process with Experiential Impact Analysis (EIA) takes as input two versions (N, N+1) of a model.

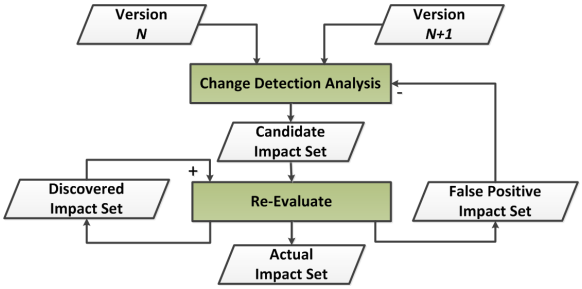

Fig. 4. Re-evaluation process with Experiential Impact Analysis. The Actual Impact Set is finally the re-evaluated set of software artifacts.

The CDA detects all model differences between both versions. In the example, the structure of the model between both version remained unchanged. Only the the content of node E1 has changed. The resulting Candidate Impact Set (CIS) is then: CIS = {E1}. As can be seen, in this process the TIA is omitted: the re-evaluation is accomplished directly after the CDA. This leads to two consequences: first, the number of false positives is much lower (compare Table I and Table II), because the CIS is smaller without a TIA, which also takes into account intermediate dependencies. Second, the DIS is larger because more impacts need to be detected during the re-evaluation (compare Table I and Table II). In this example, the manually detected impacts are: DIS = {E1,E2}. This evaluation relies on Experiential Impact Analysis (EIA) which has been described by Kilpinen [8]. The EIA is based on expert design knowledge and review techniques: code inspections and walkthroughs. In any case, these techniques have to be performed during a security evaluation [1].

TABLE II
SAMPLE SETS FOR THE PROCESS WITH EXPERIENTIAL IMPACT ANALYSIS. THE CORRESPONDING EXAMPLE IS DESCRIBED IN SECTION III-B.

	Set	**Output Of**
Candidate Impact Set	{E1}	CDA
Discovered Impact Set	{E2}	re-evaluation
False Positive Impact Set	{}	re-evaluation
Actual Impact Set	{E1,E2}	re-evaluation

It can be argued that the effort for EIA is expensive because it has to be conducted manually. Thus, a major issue in this process is to keep the cost of detecting the DIS low. This cost can be kept low, if the evaluators are involved early and

regularly. Thus, they can build up knowledge of the possible impact between software artifacts. This ensures that the EIA can be accomplished efficiently if each evaluator works on the same but narrow set of software modules.

C. Comparison of the Processes

It is an important observation here that the DIS and the FPIS are both discovered manually, whereas the CIS can be detected automatically with tools. As shown in (5) the total effort of both processes calculates:

$$Effort_{total} = Effort(DIS) + Effort(FPIS) \quad (5)$$

As can be seen, the number of DIS and FPIS are a major factor for selecting an appropriate process. Nevertheless, there are more parameters which influence such a selection. The most important factors are listed in Table III.

TABLE III
COMPARISON OF THE PROCESSES WITH TRACEABILITY IMPACT ANALYSIS AND EXPERIENTIAL IMPACT ANALYSIS.

	TIA Process	**EIA Process**
DIS	SMALL	LARGE
FPIS	LARGE	SMALL
Presumed Experience	LOW	HIGH
Iteration Cycle	Months/Years	Weeks

The process with Traceability Impact Analysis (TIA) calculates a large Candidate Impact Set because the number of traces and dependencies is usually high (see Table I). Thus, the number of direct plus indirect possible impacts soars. On the one hand, a large CIS reduces the likelihood of manually detected impacts because all direct and indirect traces have still been resolved by the TIA (see Table I). On the other hand, a large CIS will contain many false positives (see Table I and also [7]) which need to be detected manually. For human beings it is easier to falsify a visible impact than to detect an invisible impact, if no experience exists. For long time intervals between two iterations, usually no experience of the software under evaluation can be presumed. Thus, a process which detects many candidate impacts is more appropriate in this case.

The process with EIA detects more impacts via the re-evaluation process (DIS, see Table II). Fewer false positives are detected, because the number of candidate impacts is smaller without a TIA (see Table II). This process fits better for an iterative evaluation with short intervals between the re-evaluation activities. Its strength is to build up knowledge of the discovered impact set which makes the re-evaluation efficient. Although there are several iterations of re-evaluation, only one certificate is issued, in the end. In the following we will show a tooling that can assist a process with EIA.

IV. TOOL SUPPORT FOR AUTOMATION OF THE CHANGE DETECTION ANALYSIS

In order to support an agile evaluation process with tooling we implemented a Change Detection Analysis. The CDA for the security model takes as input two versions and delivers a list of all differences. For example, the CDA lists all changed source code and test entities. This has the benefit that the unchanged parts do not have to be taken into account for a re-evaluation. Basically, such a CDA for models is not new. Our contribution here is to show how such a tool can be built with existing open source software. Moreover, we faced some issues because we work with large models. These issues suggest a careful design of the model structure: each node within a model shall be tagged with a unique identifier.

A. Basic Requirements for the Tooling

First of all, the *diff engine* shall work as a standalone and independently of other proprietary tools. Unfortunately, many modeling tools do not support sophisticated model-diff engines, at present. Such a diff engine shall accept a standard format as input. The produced a diff report shall be in a format which can be easily read by humans and software. Second, we must handle large models with several thousands of artifacts. Comparing them with an inefficient algorithm is not feasible. Thus, we strive to utilize a generic diff engine which can be extended by customizable algorithms. Third, the diff engine shall be able to deal with different meta-models without much refactoring of the algorithms.

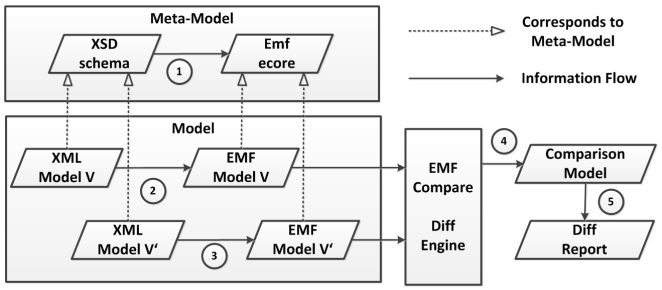

Fig. 5. Process and tooling for a Change Detection Analysis.

B. Basic Tool and Format Overview

The technical process for the CDA is shown in Fig. 5. First (1), the XSD schema[2] is imported to the EMF[3] framework and is then an EMF *ecore* meta-model. Such a meta-model describes the structure and properties of an EMF model. Then (2 and 3) the XML models are imported to the EMF framework and are then represented as EMF models. The EMF compare engine takes the two models as input and computes (4) a *comparison model*. The comparison model can be used to generate a customized diff report. This diff report is then stored in a file and contains all *add*, *delete* and *update* operations and the unique identifiers of the changed elements. The diff report can be accessed from other tools and GUI's.

[2]http://www.w3.org/XML/Schema.html
[3]http://www.eclipse.org/modeling/emf/

C. EMF Compare Customization

We identified *EMF compare*[4] as a highly customizable and extensible tool. It is open souce and can be embedded in java applications. EMF compare can be used without Eclipse in a standalone fashion. For this purpose only the appropriate *EMF compare* and *EMF ecore* libraries need to be included to a java application. Actually, we experienced some runtime issues because we were differencing large models with many differences. This is due to the standard match algorithm of EMF compare. A match engine is part of the difference engine and determines which elements of the old and new model correspond to each other. Basically, for this purpose, a similarity metric is computed for each pair of elements. The pair with the highest similarity metric is then a matched pair. Apparently, this algorithm performs poorly for models with a high number of elements. Although some optimizations of the pairwise comparison have been implemented (see [9][10]) they perform well under the assumption that not many differences exist between two versions of a model.

This assumption is not true in our case because we compare large models at a time interval of several weeks (an agile iteration). Thus, we decided to implement a match algorithm based on a comparison of unique identifiers which is much faster because no pairwise comparison is needed. EMF compare provides the possibility to utilize such a matching algorithm.

V. Related Work

The Assert4SOA[5] project is concerned with the issue of certifying software which is composed of several services. It is difficult to ascertain trust in heterogeneous software in a software ecosystem. The traditional certifying approaches do not take into account such heterogeneous systems. This project deals mainly with the issue of composing evidence for security for such systems. An approach for Common Criteria Certification of such systems is taken into account.

The EURO-MILS[6] project strives to exploit virtualization techniques in order to enable a separation between different levels of security for networked embedded systems. If an appropriate separation between highly secure software and low security software is possible, only the highly secure software needs to be certified which makes the approach more cost effective. In addition to the virtualization also the communication between the separated software entities needs to be taken into account.

The SecureChange[7] project deals with supporting security evaluation during the evolution of the software system at all levels of the software development process. A focus of this project is to focus on the delta between software releases in order to concentrate only on the changed software artifacts. Tools and processes have been developed in order to meet the mentioned objectives.

Jürjens [11] extends the UMLSec (UML) profile with annotations, so that model evolutions can be registered in the model. The original UMLSec extension can verify the model for specific security properties. The UMLSecCh can use change annotated models and compute a difference model from it. The verification is then applied to the difference model. It has been shown that such an incremental verification of security properties is much more efficient (in terms of calculation time) than a full model verification.

[4]http://www.eclipse.org/emf/compare/

[5]http://www.assert4soa.eu/

[6]http://www.euromils.eu/

[7]http://www.securechange.eu/

VI. Conclusion

Today's software systems are developed with changing requirements regarding functionality and security. Moreover, fast delivery is an important issue which is impacted by the duration of development and security evaluation. We deduced a conceptual background for an agile security evaluation which allows the management of changing requirements and fast time-to-market constraints. Moreover, this evaluation approach provides early feedback regarding the security concept of a product and thus avoids late and costly refactoring. We described a process which utilizes Change Detection Analysis and Experiential Change Impact analysis to improve such an iterative approach. In addition, we described an implementation of a change detection analysis for model-based security requirements and design.

Acknowledgment

Project partners are NXP Semiconductors Austria GmbH and TU Graz. The project is funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the FIT-IT contract FFG 832171. The authors would like to thank pure systems GmbH for support.

References

[1] *Common Criteria. Common Criteria for Information Technology Security Evaluation - Part 1-3. Version 3.1 Revision 3 Final (July 2009).*

[2] A. Cockburn, *Agile Software Development*. Pearson Education, Oct. 2006.

[3] M. Cohn, *Succeeding with Agile*. Pearson Education, Oct. 2009.

[4] K. Beznosov and P. Kruchten, "Towards agile security assurance," in *NSPW*, 2004, pp. 47–54.

[5] K. Altmanninger, P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, "Why model versioning research is needed!? an experience report," in *Proceedings of the MoDSE-MCCM 2009 Workshop@ MoDELS*, 2009.

[6] *Common Criteria. Reuse of Evaluation Results and Evidenc (October 2002).*

[7] S. A. Bohner, "Extending software change impact analysis into COTS components," in *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, 2002, pp. 175–182.

[8] M. S. Kilpinen, C. M. Eckert, and P. J. Clarkson, "The emergence of change at the interface of system and embedded software design," in *Conference on Systems Engineering Research, Hoboeken, NJ*, 2007.

[9] S. Wenzel, "Unique identification of elements in evolving software models," *Software & Systems Modeling*, vol. 13, no. 2, pp. 679–711, 2014.

[10] C. Brun and A. Pierantonio, "Model differences in the eclipse modeling framework," *UPGRADE, The European Journal for the Informatics Professional*, vol. 9, no. 2, pp. 29–34, 2008.

[11] J. Jürjens, L. Marchal, M. Ochoa, and H. Schmidt, "Incremental security verification for evolving UMLsec models," *Modelling Foundations and*, pp. 1–17, Jan. 2011.