Beyond One-Shot Security*

Jan Jürjens[1,2], Kurt Schneider[3]

[1] Software Engineering, Department of Computer Science, TU Dortmund, Germany, http://jan.jurjens.de
[2] Fraunhofer ISST, Germany
[3] Fachgebiet Software Engineering, Fakultät für Elektrotechnik und Informatik, Leibniz Universität Hannover, ks@inf.uni-hannover.de

Abstract. Security in long-living information systems requires an on-going and systematic evolution of knowledge and software for its protection. We present work towards developing techniques, tools, and processes that support security requirements and design analysis techniques for evolving information systems in order to ensure "lifelong" compliance to security requirements. We build on the security requirements & design approach SecReq developed in previous joint work. As a core feature, this approach supports reusing security engineering experience gained during the development of security-critical software and feeding it back into the development process. We develop heuristic tools and techniques that support elicitation of relevant changes in the environment. Findings will be formalized for semi-automatic security updates. During the evolution of a long-living information system, changes in the environment will be monitored and translated to adaptations that preserve or restore its security level.

1 Introduction

Information systems are exposed to constantly changing environments which require constant updating. Software "ages" not by wearing out, but by failing to keep up-to-date with its environment. Security is an increasingly important quality aspect in modern information systems. At the same time, it is particularly affected by the above-mentioned risk of "software ageing". When an information system handles assets of a company or an organization, any security loophole can be exploited by attackers. Advances in knowledge and technology of attackers are part of the above-mentioned environment of a security-relevant information system. Outdated security precautions can, therefore, permit sudden and substantial losses. Security in long-living information systems, thus, requires an on-going and systematic evolution of knowledge and software for its protection. Our objective is to develop techniques, tools, and processes that support security requirements and design analysis techniques for evolving information

* This work is supported by the DFG project SecVolution (SPP 1593 “Design For Future – Managed Software Evolution”) and dedicated to Martin Glinz on the occasion of his 60th birthday.

systems in order to ensure "lifelong" compliance to security requirements. We will build on the security requirements & design approach SecReq developed in previous joint work. As a core feature, this approach supports reusing security engineering experience gained during the development of security-critical software and feeding it back into the development process. We will develop heuristic tools and techniques that support elicitation of relevant changes in the environment. Findings will be formalized for semi-automatic security updates. During the evolution of a long-living information system, changes in the environment will be monitored and translated to adaptations that preserve or restore its security level.

There are several research challenges for our vision of co-evolving knowledge and software models. These challenges are driven by foreseeable changes in the environment of a security-relevant information system: New regulations require compliance; the source of new regulations (standards organizations, law-makers, interest groups etc.) are usually well-known. Moreover, failures will be detected in the construction (i.e. models or code) of the information system itself. When it comes to new or changing technologies and knowledge, it is more difficult to anticipate where those changes might originate. For example, faster computers can make it easier to decode simplistic encodings. New approaches like rainbow tables [Oec03] can compromise password encoding schemes that were considered sufficient before. A rainbow table is a precomputed table for reversing cryptographic hash functions. This technique, published in 2003, is used for cracking password hashes and has be successfully applied to various systems. Challenge 1 is the elicitation and management of such knowledge sources.

When knowledge sources are identified, information on new knowledge and its potential impact on security must be elicited. This analysis needs to reach a level of detail that allows conceiving counter-measurements and updating of models during co-evolution. Challenge 2 addresses both technical and human aspects. Knowledge acquisition in general is a difficult and often time-consuming endeavor [NT95]. In this context, identifying relevant bearers of knowledge and getting the security-relevant fraction of their new insights will not be achieved by classical interviews or questionnaires: On the one hand, new techniques are required to make implicit findings and tacit knowledge explicit – without requiring a lot of additional effort. This is a well-known precondition for collaboration of knowledgeable experts in a knowledge or experience-sharing initiative [Sch09b]. On the other hand, resulting UMLsec models are supposed to be formal enough for deriving code. Raw input from elicitation must be turned into UMLsec models. Since changes in the environment are the reason for updating an information system, an *environment model* should play an intermediary role. Incoming information on changes need to be represented in that model. On the back-end, represented changes trigger respective changes in the software models and their security aspects by means of co-evolution. In addition to this conceptual and technical challenge, the knowledge in the en-

vironment model, requirements, and UMLsec models must be maintained and managed.

The ultimate goal is to preserve the security of an information system by adapting software models through the use of external and internal knowledge sources.

2 The SecReq Approach

There is a significant amount of research in security-enhanced UML modeling [JW01,Jür01,Jür05,JS07a,HJ08,JSB08,WPP+09] and related foundations [Jür00,Jür02]. These models provide the opportunity to tag UML models with specific markers. As a consequence, security-related properties at these markers can be demonstrated or monitored automatically. Further previous work includes requirements engineering for evolving systems [BDLS06,SMP+10], [Sch11], in particular in the early phases of requirements elicitation and analysis [Sch96,KSS09,BSK10]. Information flow in software projects was investigated [Sch04,SL05], [Sch07,Sch08a], and applied in industry [SSLF07]. A graphical modeling notation to capture, improve, and even for designing new information flows was developed [SSK08].

In previous work, the research groups of the two authors have collaborated with the goal of identifying security-related requirements and statements very early during the general requirements elicitation process. Heuristic elicitation techniques were coupled with UMLsec, which supports the construction of secure systems [HIK+10,KHS+11,SKH+12].

Overview. In principle, vague and informal input from various stakeholders (left) was elicited and filtered by the heuristic requirements assistant (HeRA) during the elicitation activity. Fig. 1 shows the flow of information from stakeholders through two activities (requirements elicitation and construction) to a secure system. The figure shows how project-specific information flows from requirements elicitation to construct system, while experience flows back to improve the elicitation activity. Resulting requirements were documented (center) and had been improved by identifying and discussing potentially security-relevant statements. This increased awareness for security made them better suited for tagging in UMLsec models (right), and finally secure systems. As stated above, UMLsec models are the interface of our work to the work of many other researchers; implementation of a secure system is beyond our scope.

Our collaboration was inspired by the vision of establishing a feedback loop from construction to elicitation. The gray arrow on top of Fig. 1 indicates how insights and experience were used in subsequent elicitation activities during subsequent projects. This principal structure has been instantiated and substantially refined since then.

UMLsec. Although several approaches for model-based secure software engineering exist, few of these include automated tools for formally verifying the

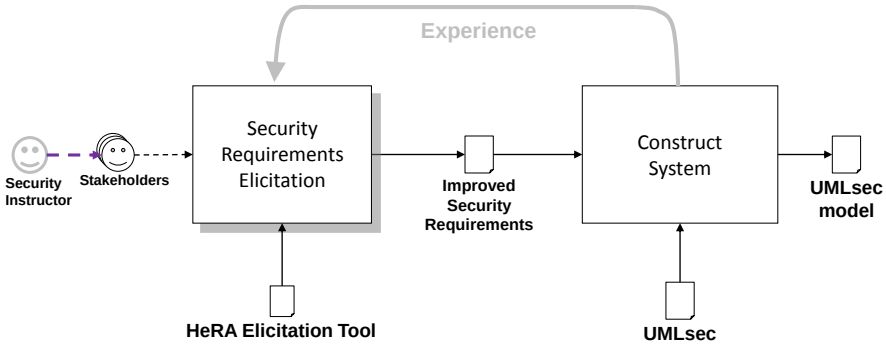

Fig. 1. Overview of information flows in our SecReq approach.

models against the security requirements. In this project, we focus on one such approach (based on the UML extension UMLsec [Jür05]) that does offer automated tools to verify UML models against security requirements, because giving such tools to practitioners in industry significantly improves the practical impact of the scientific results that will be achieved. The *UMLsec* tool [JS07b] supports the analysis of the security aspects expressed in the security extension UMLsec [Jür05] of the Unified Modeling Language (UML). The tool mainly focuses on the verification of the most important security requirements, which can be directly used in the model, together with their formal definitions.

In the UML extension UMLsec, [Jür05], recurring security requirements (such as secrecy, integrity, authenticity and others) and security assumptions on the system environment, can be specified within a UML specification as annotations. This way we can encapsulate knowledge on prudent security engineering and make it available to developers who may not be security experts. The UMLsec extension is given in form of a UML profile using the standard UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate the security requirements and assumptions. *Constraints* give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics of the used fragment of UML. The security-relevant information added using stereotypes includes security assumptions on the physical level of the system, security requirements related to the secure handling and communication of data, and security policies that system parts are supposed to obey.

The information flow interface of UMLsec is depicted in Fig. 1 on the right hand side. UMLsec supports the construction activity. The input of that activity is an improved requirements specification with security-relevant parts being identified and marked. The ultimate outcome is supposed to be a secure sys-

tem. For the purpose of this proposal, the boundary is more precisely specified as a set of security-enhanced UMLsec models.

UMLsec by itself does not address the issue of eliciting security requirements and marking statements as potentially security-relevant. This input must be provided by other sources or techniques.

Heuristic Elicitation as a By-Product. Previouse work investigates the elicitation, analysis [KL08,KLM09,BSK10], and flow of requirements and rationale [SSK08,SKS09,Sch06,SKSZ11] in software organizations [LS97,BDLS06], [HIK+10,Sch07].

Elicitation is particularly difficult: A lot of the assumptions, knowledge and experience associated with a requirement or a rationale are tacit [Pol66]. People are unaware of their existence and importance, and they cannot express or document them. Therefore, techniques or tools must trigger reflection-in-action in order to create (desired) breakdowns [Sch83]. When knowledge workers are interrupted, they gain the opportunity to become aware of tacit knowledge they were just about to apply unconsciously. However, there is a tension between overlooking tacit knowledge and interrupting too much. In addition, holders of relevant informations may not be willing or able to spend extra effort on documenting it for the benefit of others. As Grudin pointed out for CSCW systems, benefit and effort must be balanced for all participants [Gru87]. Therefore, techniques have been conceived for capturing as much relevant information as possible as a by-product.

As early as 1996, Schneider presented a first example of such a specific tool taking the above-mentioned tension into account: FOCUS served to capture rationale from a demo and subsequent code explanations was presented [Sch96]. Demos and explanations were recorded and connected with a trace of executed methods in Smalltalk code. That trace was then used to guide explanations. After several demos, a network of traces, explanations, and audio records of what had been said enriched the raw code with a network of knowledge and rationale. The concept of FOCUS was reimplemented in Java and the Eclipse IDE. The respective approach of capturing valuable information as a by-product of other activities was generalized into the “By-Product Approach” [Sch09a]. The approach provides guidelines for constructing tools and techniques that would collect rationale as a by-product of an activity which is carried out anyway (like a demo in FOCUS). This approach was applied, e.g., by combining use case writing with manual mockups [Sch07]. However, each source of information or knowledge and each specific situation calls for a new idea to instantiate and optimize the by-product approach.

An important by-product of writing a specification can be a first heuristic check of its plausibility and formal consistency. Mistakes and misunderstandings during elicitation can lead to severe consequences in later phases of a project. Therefore, heuristics for assessing and evaluating early manifestations of requirements are monitored for identifying symptoms of problems [Sch08b]. In a specification, for example, incomplete grammar, excessively long

sentences, or complex structure of requirements may lead to misunderstandings. At deeper levels of analysis, poor use of domain terminology or even patterns of semantic flaws can be identified. Due to the heuristic nature of this analysis, errors cannot be fully avoided. In turn, the approach provides very early feedback. In most contexts, findings should be discussed by stakeholders or experts. Their attention can be focused to a subset of a lengthy requirements document that is just being written.

As Fig. 1 (left hand side) illustrates, the input for heuristic analysis can even be informal, like a meeting of stakeholders, several emails, or a phone call. This is denoted by the face symbol representing fluid information. Fluid information will be defined below; in short, un-documented information is fluid, although there are fine theoretical considerations [SS12]. Tools like HeRA (Heuristic Requirements Assistant) [KLM09] were developed to apply heuristic rules to a given requirement. HeRA contains requirements editors so that a statement or requirement can be analyzed when it is first documented, e.g. during a stakeholder meeting. HeRA has been applied and evaluated in both industrial and educational software projects [Kna10]. We were able to show that these concepts lead to better [KF07] and more complete [KL08] requirements documentation, while supporting organizational learning [KSS09].

Information Flow Modeling in FLOW. In previous work, we model, analyze, and improve information flows in software projects. The core concepts of the approach are: So-called *fluid* information is modeled along with *solid* information. *Solid* is characterized as information that is long term accessible, repeatable, and comprehensible by third parties. Any non-solid information is called *fluid*. Experience is considered a specific type of information that deserves specific attention. Modeling must be easy and encourage discussions. For details and discussions see [SSK08,SS12].

A notation for visualizing information flows was designed to meet the following criteria:

- The notation should be easy to use on whiteboards, in general drawing tools like PowerPoint, and in custom-built editors. In particular, the notation should contain only few symbols and details. It must not distract from the information flow focus.
- The notation must explicitly visualize and distinguish solid and fluid information and flows.
- Experience occurs as a specific type of information. Experience often acts as catalyst on other information flows. Despite its prominent role, experience is rarely considered explicitly. In FLOW, however, experience is taken very seriously. Ordinary project information and experience must be distinguishable at first glance.

The resulting syntax is summarized in Table 1. This notation has been used to model information flow in various figures of this proposal, such as Fig. 1 and Fig. 2. It was also used continuously in our joint research since 2008.

– A document symbol represents solid information. A human face is the symbol for a store of fluid information, since people have a lot of information in their minds. Multiple document or face symbols refer to one or more stores (groups) of the same document type.
– Flows are represented by arrows. All flows originating from a solid store are solid. All flows originating from a fluid store are fluid. Flows originating from an activity can be either solid or fluid: A model builder can express an intention or assumption by using fluid or solid style.
– Labels refer to individuals or roles, depending on the purpose of a model. Labels on arrows can be used to highlight specific types of information flowing.
– Experience is depicted in gray. Requirements and other information is depicted in black.
– The activity symbol (rectangle) is used as follows: Incoming and outgoing flows are connected to the rectangle on the left or right sides. Controlling flows are connected at the top and tools at the bottom of the rectangle. This symbol serves for hierarchical decomposition of FLOW models. They can hide details in a black box, which supports scaling of models.

Table 1. FLOW notation symbols according to Schneider et al. [SSK08].

State	*Information Store*	*Information Flow*	*Experience Flow*	*Activity*
solid	<Document name>	<Information typ> (optional)	<Experience> (optional)	<Control, support, by-product> <Name of Activity> <In- / out-going information>
fluid	<Person>	<Information typ> (optional)	<Experience> (optional)	

Research on Security Requirements Elicitation and Engineering. The European Telecommunications Standards Institute (ETSI) is a major standardization organization within the telecommunications domain. It was responsible for the standardization of GSM, UMTS and LTE (2G, 3G and 4G). ETSI is member-driven; members include ISP, smart card providers, network providers, and others and spans across Europe, Asia and the US. One of the problems at ETSI is the shortage of security experts. Since many projects are initially not considered security-relevant, little attention is paid to those projects. After a while, a newly designed system can be linked or integrated with additional features that require security engineering.

The authors wanted to mitigate the shortage of security expertise by systematically reusing experience. The approach was initiated by sketching and refining the intended flow of requirements and security experience using FLOW. The focus was to identify potentially security-relevant requirements and statements early in the process. This would allow to assign security experts efficiently.

The initial approach was called *SecReq* and presented in [HIK+10]. SecReq combines three distinctive techniques to support security requirements elicitation: (1) Common Criteria [Org07] and its underlying security requirements elicitation and refinement process, (2) the HeRA tool [KLM09] with its security-related heuristic rules, and (3) the UMLsec tool set [Jür05] for security analysis and design. In [KHS+11] we focused on HeRA's ability to automatically detect security-relevant requirements based on Bayesian classifiers.

The vision of SecReq was to assist all steps in security requirements elicitation, as well as providing mechanisms to trace security requirements from high-level security statements (security objectives) to rather secure design [HIK+10]. The approach aims at bridging the gap between security best practices and the lack of security experience among developers and designers.

Heuristic critiques can be used to *identify security relevant requirements*, e.g. based on Bayesian classifiers [KHS+11]. Once identified, a wizard guides analysts through a refinement process. The wizard helps to document requirements according to Common Criteria. In addition, it guarantees that *security requirements are specified in sufficient quality* to create a secure design with the UMLsec tool set.

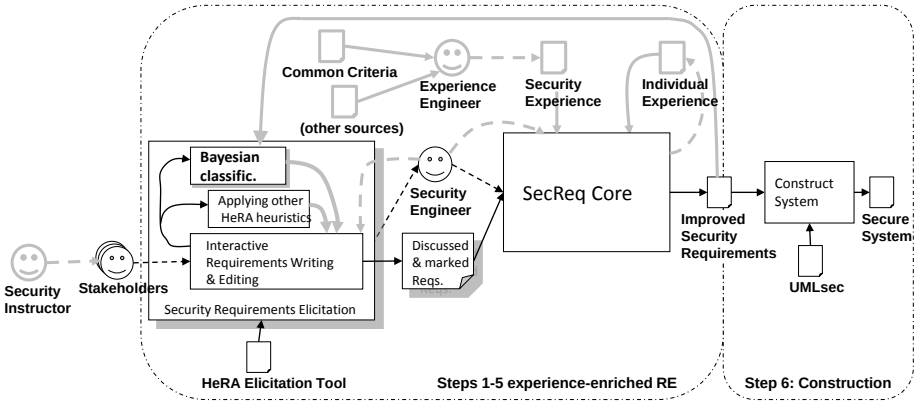

Fig. 2. Complete model of information flows in SecReq.

It is important to note that the narrow focus of SecReq allowed us to construct and optimize appropriate heuristic elicitation techniques designed for reusing experience. E.g., Bayesian classifiers were trained with final specifications of previous projects, where security-related issues had been identified

at last. The Bayesian classifiers represent solid expertise in identifying security-related requirements [KHS+11]. The FLOW model in Fig. 2 shows the sophisticated information and experience flows at this stage. There are two intertwined, but different challenges: (1) Designing the information flow overview in FLOW and (2) conceiving the techniques and tools to support it, as discussed in [SKH+12]. In SecReq, we were able to demonstrate the feasibility of this general approach when applied to one specific aspect of security engineering.

3 Open Challenges

Extending the scope and goal to other types of information, sources of incoming knowledge and tacit security knowledge poses substantial new challenges: How can the knowledge of attackers be elicited and considered against their will? Can the SecReq mechanisms be adopted and extended to scan new legislation and regulations for security-related issues – or how can human intermediaries provide their insights efficiently? How much of a large, existing information system must be scanned and reconsidered for security impact, given a certain type of identified change? And are there any generic principles for building heuristic tools beyond the first set of recommendations provided in [Sch06]? So far, each particular idea of eliciting or validating slightly different aspects calls for substantially new and innovative tools and techniques.

Based on the state of the art and our previous work, we identify the following open challenges which remain to be solved:

- **Challenge 1:** Systematic and experience-based elicitation and management of multiple, heterogeneous knowledge sources throughout the lifecycle of a long living system.
- **Challenge 2:** Systematic analysis and optimization of how new knowledge affects security of long-living software systems.
- **Challenge 3:** Maintaining a consistent database of security requirements and security-relevant environment knowledge during evolution of a long-living information system.

Acknowledgements The collaboration with the other members in the SecVolution project is gratefully acknowledged: Stefan Gärtner, Stephan Kiesling and Olga Liskin from Leibniz Universität Hannover, and Thomas Ruhroth and Sven Wenzel from TU Dortmund.

Dedication We dedicate this paper to Martin Glinz, who is interested in many of the areas we investigate in this work:

- When Kurt Schneider first met Martin Glinz more than 20 years ago, Martin was still working at ABB in Switzerland. Back then, he participated in meetings of the Requirements Engineering SIG in Stuttgart.
- Jan Jürjens has had the pleasure of various lively discussions with Martin Glinz in the context of meetings organized by the GI Querschnitts-FA Modellierung.

– Our focus on security and security requirements is a link to Martin's famous work of non-functional requirements.
– His current interest in informal notations and sketches is another interest we share, as the FLOW notation presented above shows.

These examples illustrate the similarity in our thinking. We were both inspired by Martin's research, and by his loyalty to requirements engineering. Thank you!

References

[BDLS06] Andreas Birk, Torgeir Dingsøyr, Stefanie Lindstaedt, and Kurt Schneider, editors. *Learning Software Organisation and Requirements Engineering: The First International Workshop*. Know-Center and Graz University of Technology, 2006.

[BSK10] Olesia Brill, Kurt Schneider, and Eric Knauss. Videos vs. Use Cases: Can Videos Capture More Requirements Under Time Pressure? In Roel Wieringa and Anne Persson, editors, *Proceedings of the 16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '10)*, volume 6182 of *LNCS*, pages 30–44, Essen, Germany, 2010. Springer.

[Gru87] J. Grudin. Social evaluation of the user interface: Who does the work and who gets the benefit. In *INTERACT'87. IFIP Conference on Human Computer Interaction*, 1987.

[HIK+10] Siv Hilde Houmb, Shareeful Islam, Eric Knauss, Jan Jürjens, and Kurt Schneider. Eliciting Security Requirements and Tracing them to Design: An Integration of Common Criteria, Heuristics, and UMLsec. *Requirements Engineering Journal*, 15(1):63–93, March 2010.

[HJ08] Sebastian Höhn and Jan Jürjens. Rubacon: automated support for model-based compliance engineering. In Schäfer et al. [SDG08], pages 875–878.

[JS07a] Jan Jürjens and Pasha Shabalin. Tools for secure systems development with uml. *STTT*, 9(5-6):527–544, 2007.

[JS07b] Jan Jürjens and Pasha Shabalin. Tools for secure systems development with UML. *Int. J. Softw. Tools Technol. Transf.*, 9(5):527–544, 2007.

[JSB08] Jan Jürjens, Jörg Schreck, and Peter Bartmann. Model-based security analysis for mobile communications. In Schäfer et al. [SDG08], pages 683–692.

[Jür00] Jan Jürjens. Secure information flow for concurrent processes. In Catuscia Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2000.

[Jür01] Jan Jürjens. Modelling audit security for smart-cart payment schemes with uml-sec. In Michel Dupuy and Pierre Paradinas, editors, *SEC*, volume 193 of *IFIP Conference Proceedings*, pages 93–108. Kluwer, 2001.

[Jür02] J. Jürjens. *Principles for Secure Systems Design*. PhD thesis, Oxford University Computing Laboratory, 2002.

[Jür05] J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, 2005.

[JW01] Jan Jürjens and Guido Wimmel. Formally testing fail-safety of electronic purse protocols. In *ASE*, pages 408–411. IEEE Computer Society, 2001.

[KF07] E. Knauss and T. Flohr. Managing Requirement Engineering Processes by Adapted Quality Gateways and critique-based RE-Tools. In *Proceedings of Workshop on Measuring Requirements for Project and Product Success*, Palma de Mallorca, Spain, November 2007. in conjunction with the IWSM-Mensura Conference.

[KHS+11] Eric Knauss, Siv Houmb, Kurt Schneider, Shareeful Islam, and Jan Jürjens. Supporting Requirements Engineers in Recognising Security Issues. In Daniel Berry and Xavier Franch, editors, *Proceedings of the 17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '11)*, LNCS, Essen, Germany, 2011. Springer.

[KL08] Eric Knauss and Daniel Lübke. Using the Friction between Business Processes and Use Cases in SOA Requirements. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC), Workshop on Requirements Engineering For Services*, pages 601–606, Turku, Finland, 2008.

[KLM09] E. Knauss, D. Lübke, and S. Meyer. Feedback-Driven Requirements Engineering: The Heuristic Requirements Assistant. In *International Conference on Software Engineering (ICSE'09), Formal Research Demonstrations Track*, pages 587 – 590, Vancouver, Canada, 2009.

[Kna10] Eric Werner Knauss. *Verbesserung der Dokumentation von Anforderungen auf Basis von Erfahrungen und Heuristiken*. Cuvillier Verlag, Göttingen, Germany, 2010. Phd Thesis.

[KSS09] E. Knauss, K. Schneider, and K. Stapel. Learning to Write Better Requirements through Heuristic Critiques. In *Proceedings of 17th IEEE Requirementes Engineering Conference (RE 2009)*, Atlanta, USA, 2009.

[LS97] Stefanie N. Lindstaedt and Kurt Schneider. Bridging the Gap between Face-to-Face Communication and Long-term Collaboration. In *Proceedings of GROUP 97*, Phoenix, USA, Nov 1997. ACM.

[NT95] Ikujiro Nonaka and Hirotaka Takeuchi. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, Oxford, 1995.

[Oec03] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer Berlin / Heidelberg, 2003.

[Org07] International Standardization Organization. ISO 15408:2007 Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 2, CCMB-2007-09-001, CCMB-2007-09-002 and CCMB-2007-09-003, September 2007.

[Pol66] M. Polanyi. *The Tacit Dimension*. Doubleday, Garden City, NY, 1966.

[Sch83] D.A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, 1983.

[Sch96] Kurt Schneider. Prototypes as Assets, not Toys. Why and How to Extract Knowledge from Prototypes. In *18th International Conference on Software Engineering (ICSE-18)*, pages 522–531, Berlin, Germany, 1996.

[Sch04] Kurt Schneider. A Descriptive Model of Software Development to Guide Process Improvement. In *Conquest*, Nürnberg, Germany, 2004. ASQF.

[Sch06] Kurt Schneider. Rationale as a By-Product. In A. H. M. Dutoit, I. Mistrik, and Barbara Paech, editors, *Rationale Management in Software Engineering*, pages 91–109, Berlin, Heidelberg, 2006. Springer.

[Sch07] K. Schneider. Generating Fast Feedback in Requirements Elicitation. In *Requirements Engineering: Foundation for Software Quality (REFSQ 2007)*, 2007.

[Sch08a] Kurt Schneider. Can we talk? Communication vs. Documentation in Software Projects. In *Keynote at the 4th World Congress for Software Quality (WCSQ)*, Bethesda, Maryland, USA, 2008. American Society on Quality (ASQ).

[Sch08b] Kurt Schneider. Improving Feedback on Requirements through Heuristics. In *Proceedings of 4th World Congress for Software Quality (4WCSQ)*, 2008.

[Sch09a] Kurt Schneider. A Spectrum of Languages for Exploring Requirements in IT Ecosystems. submitted to RE09, 2009.

[Sch09b] Kurt Schneider. *Experience and Knowledge Management in Software Engineering*. Springer-Verlag, 2009.

[Sch11] Kurt Schneider. Focusing Spontaneous Feedback to Support System Evolution. In *Proceedings of IEEE 19th International Requirements Engineering Conference (RE'11)*, pages 165–174, Trento, Italy, 2011. IEEE.

[SDG08] Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors. *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*. ACM, 2008.

[SKH+12] Kurt Schneider, Eric Knauss, Siv Houmb, Shareeful Islam, and Jan Juerjens. Enhancing Security Requirements Engineering by Organizational Learning. *Requirements Engineering Journal (REJ), special issue on REFSQ'11*, 2012. accepted for publication (12. Nov 2011).

[SKS09] Kai Stapel, Eric Knauss, and Kurt Schneider. Using FLOW to Improve Communication of Requirements in Globally Distributed Software Projects. In *Workshop on Collaboration and Intercultural Issues on Requirements: Communication, Understanding and Softskills (CIRCUS '09)*, Atlanta, USA, November 2009.

[SKSZ11] Kai Stapel, Eric Knauss, Kurt Schneider, and Nico Zazworka. FLOW Mapping: Planning and Managing Communication in Distributed Teams. In *Proceedings of 6th IEEE International Conference on Global Software Engineering (ICGSE '11)*, pages 190–199, Helsinki, Finland, 2011.

[SL05] Kurt Schneider and Daniel Lübke. Systematic Tailoring of Quality Techniques. In *World Congress of Software Quality 2005*, volume 3, 3, 2005.

[SMP+10] Kurt Schneider, Sebastian Meyer, Maximilian Peters, Felix Schliephacke, Jonas Mörschbach, and Lukas Aguirre. Feedback in Context: Supporting the Evolution of IT-Ecosystems. In *PROFES*, pages 191–205, 2010.

[SS12] Kai Stapel and Kurt Schneider. Managing Knowledge on Communication and Information Flow in Global Software Projects. *Expert Systems - Special Issue on Knowledge Engineering in Global Software Development*, 2012. submitted for consideration to.

[SSK08] Kurt Schneider, Kai Stapel, and Eric Knauss. Beyond Documents: Visualizing Informal Communication. In *Proceedings of Third International Workshop on Requirements Engineering Visualization (REV 08)*, Barcelona, Spain, 2008.

[SSLF07] K. Stapel, K. Schneider, D. Lübke, and T. Flohr. Improving an Industrial Reference Process by Information Flow Analysis: A Case Study. In *Proceedings of PROFES 2007*, volume 4589 of *LNCS*, pages 147–159, Riga, Latvia, 2007. Springer-Verlag Berlin Heidelberg.

[WPP+09] C. Murray Woodside, Dorina C. Petriu, Dorin Bogdan Petriu, Jing Xu, Tauseef A. Israr, Geri Georg, Robert B. France, James M. Bieman, Siv Hilde Houmb, and Jan Jürjens. Performance analysis of security aspects by weaving scenarios extracted from uml models. *Journal of Systems and Software*, 82(1):56–74, 2009.