Reusing Security Solutions: A Repository for Architectural Decision Support

Stefanie Jasser Matthias Riebisch

Department of Informatics, University of Hamburg
Hamburg, Germany
{jasser, riebisch}@informatik.uni-hamburg.de

ABSTRACT

Today, the interplay of security design and architecting is still poorly understood and architects lack knowledge about security and architectural security design. Yet, architectural knowledge on security design and its impact on other architectural properties is essential for making right decisions in architecture design. Knowledge is covered within solutions such as architectural patterns, tactics, and tools. Sharing it including the experience other architects gained using these solutions would enable better reuse of security solutions.

In this paper, we present a repository for security solutions that supports architectural decisions including quality goal trade-offs. Its metamodel was adapted to special demands of security as a quality goal. The repository supports architecture decisions not only through populating approved solutions but through a recommender system that documents knowledge and experiences of architecture and security experts. We provide a case study to illustrate the repository's features and its application during architecture design.

Keywords

Security by Design; Reusing Security Solutions; Secure Architecture; Software Architecture; Secure Software Development; Security Engineering

1. INTRODUCTION

Designing secure software architectures is a complex task. Worse, software engineering methodologies today still lack support for integrating security with other architectural properties. That especially holds true for the integration of security and architectural aspects during the design phase. One problem is the lack of knowledge about security and security enhancing methodologies and technologies (e. g. security pattern) among software engineers. Such security enhancing solutions often have a negative effect on other quality properties such as usability or availability, which might be important for a specific project. In such a case, architects are forced to trade quality goals off against each other during decision making. If an architect's decisions could benefit from other architects' knowledge and from the experiences of experts this would reduce the risk and improve the quality of those decisions. However, reuse is limited through the accessibility of security enhancing solutions and knowledge about their impact on a system.

We aim for better reuse of security mechanisms by providing a repository that supports architects in making design decisions. Similar to most architectural methodology approaches, the repository contains both abstract solutions – e. g. design patterns, architectural design principles or reference architectures, and technical components. These include libraries, products (OS, middleware, DBMS, . . .), or components like code snippets. Yet, architectural styles, patterns etc. do not cover all aspects of knowledge since a formal description is missing. Omitted aspects might be: knowledge about a solution's impact on a specific quality property, the contribution of a solution to functional properties, preconditions for a solutions' application, and the skills needed to implement a solution. Our approach provides security engineering knowledge including that informal knowledge as a part of the software architect's repository. In this way we reduce the complexity of architectural decision making, because architects can search for solutions by specifying their needs as well as constraints that result from the system's environment.

Solutions and components in the repository are rated regarding their contribution to the aimed solution in terms of the overall quality goals of a software system. Therefor, each solution can be evaluated through a recommender system that allows architects to share their experiences with this solution's application. Those recommendations should especially contain information on a solution's implications to non-functional requirements to facilitate the aforementioned trade-off decisions.

To illustrate our approach, we provide a case study of an enterprise resource planning system. To enable business processes across company boundaries web services are used. Security is an important quality goal for the given system as the company's customers shall be allowed to access their own order status but they must not access any information on processes regarding their competitors. Yet, maintainability must not be disregarded, too.

Towards our main goal to enable reusability of security mechanisms and security engineering knowledge, the contribution of this paper consists in providing both security knowledge and solutions for architecture design covering such

ECSAW '16, November 28-December 02, 2016, Copenhagen, Denmark

ISBN 978-1-4503-4781-5/16/11. . . $15.00

DOI: http://dx.doi.org/10.1145/2993412.3007556

knowledge in a repository. Architectural design decisions are facilitated by providing information about the impact of solutions on a system's quality properties, about constraints, and about dependency relationships with other solutions and with the existing environment. Our approach utilizes other works to populate the repository and to maintain the covered information by data extracted from developer networks. Furthermore, the approach makes experiences available from other developers that are collected from user ratings. The approach is prepared for an extension to other quality attributes such as safety, availability or usability.

2. RELATED WORK

2.1 Sharing Architectural Knowledge

In the design stage architects need to make various decisions on the system. These decisions should explicitly include security-related decisions, which should not be left to developers. To make an appropriate architecture design decision an architect has to have access to architectural knowledge originating from his own experience as well as external sources [9].

In the history of software reuse, several approaches have been proposed to provide and share solutions and knowledge. Most of them deal with patterns, best practices or anti-patterns ordered in catalogues, which are presented on websites or in books. These catalogues are in danger to become outdated or deprecated soon due to the high effort or other obstacles maintaining them. They further usually do not benefit from experiences made subsequently [20]. Some well-known examples are [9, 20, 19]. Besides the mentioned abstract solutions there are various solutions that are more concrete: Libraries encapsulate complexity and knowledge and make it accessible to others, tools support approaches or indicate problems, etc. Other means of representing and disseminating knowledge especially for security are security alerts and bulletins.

Yet, in addition to those classical ways of disseminating knowledge, there are also some tools for capturing, reasoning and sharing architectural knowledge. These tools basically differ in their capability to assist an architect with the decision making process. Well-known tools are: The Process-centric Architecture Knowledge Management Environment (PAKME) [1], which allows the acquisition, retrieval and presentation of architectural knowledge. Generic patterns are provided through a library to support architecture design. The Architecture Design Decision Support System (ADDSS) introduced in [6] provides the capability to iteratively document architectural design decisions and dependencies between them. These dependencies are represented through constraint relationships. Solving new design issues is supported by a pattern repository. The Environment for Architects to Gain and Leverage Expertise (EAGLE) tool [8] is a portal that provides various services for sharing architectural knowledge. The tool focuses upon collaboration among architects. A detailed survey of architectural knowledge tools can be found in [26, 21]. While traceability is well supported by all tools, they often lack sufficient features for integrating knowledge into a user's individual context and for sharing experiences. Additionally, we are not aware of tools that meet security-related needs that go beyond patterns or tactics, but for example includes common attacks.

2.2 Modelling Architectural Knowledge

To present solutions contained in the repository there exist some approaches to model architectural security properties. While some approaches use other ADLs (e. g. in [15, 16]), most of them are based on the Unified Modelling Language (UML) [3]: UMLsec is an important extension of UML using stereotypes and tags to add security-relevant information [12, 13]. It provides automated tool support and can be used to verify systems against attacker models from threat scenarios.

Though UMLsec is a well-known approach for modelling security concerns, there are other notations based on UML: e. g. SecureUML [2], which allows to model Role Based Access Control (RBAC), and Modelling Architectural Security Concerns (MASC) [23], which makes it possible to document architectural decisions for some security engineering principles.

Besides advantages in using UML as a basis for modelling notations – which is mainly that it is the de-facto standard notation for modelling in software engineering – there are severe disadvantage, too: Firstly, approaches that add elements to the original UML notation further increase the notation's complexity. Yet, UML itself already has huge inherent complexity and is rarely used entirely or correctly. Hence, additional complexity makes the approaches even harder to use. Secondly, UML does not concentrate on modelling architectural concerns and design decisions.

2.3 Multi-Criteria Decisions

Making architectural design decisions often requires considering multiple criteria. Approaches that deal with trade-off decision making are called Multi-Criteria Decision Making or Analysis methods. These methods can be used to evaluate a set of architectural design alternatives regarding project-specific quality goals. The numbers of design alternatives and quality goals are usually assumed to be finite. The overall ranking of design alternatives then is determined by combining the rankings induced by each quality goal with the quality goal's weight. [28]

Another method used frequently is to reduce multiple criteria to a single one. Yet, we lack fair research on the consolidation of multiple criteria to a single criterion meeting the user's needs.

As a well-known approach, the Analytic Hierarchy Process (AHP) approach decomposes the problem of multi-criteria decision making into a hierarchy of quality goals. Decisions are made at each level using pairwise comparisons [18]. In [7, 27] comparative studies of decision-making techniques have been conducted.

3. A REPOSITORY FOR SECURITY SOLUTIONS

3.1 Preliminary Work

In the authors' research group there have been works towards this paper's goal. A repository for providing reusable solutions has been developed during a PhD project [4]. The repositories' classification called Goal Solution Scheme GSS enabled a specification of the solutions' contribution to quality goals. Layer III with design principles facilitates a context-independent evaluation of the solutions' contribution (see Figure 1 for an example) as design principles are context-

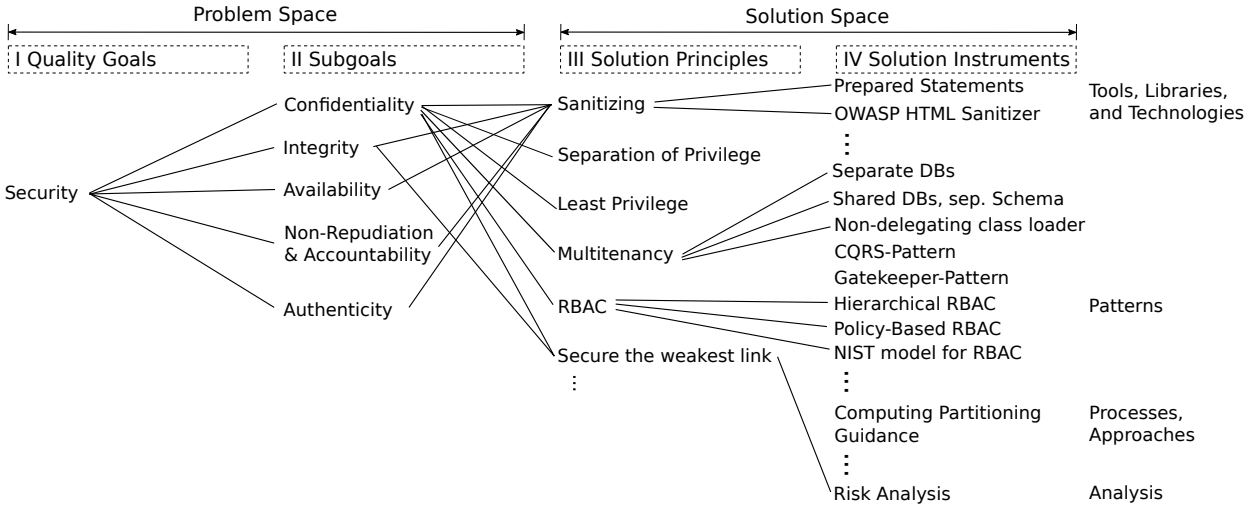

Figure 1: The Goal Solution Scheme layers with an example[1]

independent themselves. Competing and conflicting quality goals can be managed in multi-criteria decisions [17] [5]. Recommendations of solutions for developers should not include incompatible solutions. Therefor, constraints have been introduced to represent preconditions for the application of solutions, and to enable a preselection of applicable solutions before recommending them in a ranked list [14]. Reusable artefacts for system development are not limited to building blocks and abstract solutions, but process patterns such as refactorings had be considered, too. They are important for software evolution and reengineering [10]. Design decisions about technologies bear a higher risk. Therefore, they need a special support by specifying their relations and contributions [25]. Tool support has been developed to master the complexity of multiple goals, relations between solutions and their classification in repositories. We developed both an Eclipse-based and a web-based tool for managing the repository [11]. For populating a repository and for maintaining its content in times of rapid technical progress, we captured information from developers' blogs such as StackOverflow [24].

3.2 Structure of the Repository

The repository shall provide guidance during search, and it shall represent relationships between elements. As stated in our preliminary works, there are multiple search criteria, including a solution's contribution to a variety of quality goals, relationships to other solutions, properties regarding constraints, and relations to knowledge concepts. The repository contains security solutions of different types and different levels of abstraction. Examples are security principles and tactics, design patterns, frameworks, or cryptographic libraries.

There are two different basic approaches to access collected elements, by a classification or by an index. According to Shaw's recommendation [22], an index shall be preferred. Even if a classification is easier to maintain, it has several disadvantages. Firstly, the hierarchical structure of a classification is not appropriate for cross-linked items such as knowledge concepts. Secondly, a classification fails to represent multiple and heterogeneous viewpoints. Thirdly, a classification has to be maintained by humans, though there is no effective business model for such maintenance services as we know from the history of the Internet and from software component markets. An index can be maintained with much more tool support compared to a classification [22].

In our approach, we decided for a combination of both worlds. We structure repository elements through keywords that are used to generate structures in addition to a classification. The use of an ontology taken as a basis and keywords allows performing searches for appropriate solutions. Yet, keywords can be used to further expand the resulting design space by conducting similarity searches if the initial design space lacks potential design alternatives. Similarly to a classification scheme, we developed the Goal Solution Scheme (GSS) in earlier works of our research group (see Sect. 3.1). It has advantages because there are determined criteria for searching elements in a repository, such as quality properties and engineering concepts. Furthermore, the GSS represents dependency relationships of solutions to quality goals explicitly. We introduced the GSS in [5] in detail.

The GSS differentiates four layers that have bottom-up dependencies (see Figure 1 for an example): A dependency relationship indicates that a solution instrument adheres to a solution principle, i. e. it is in a way compliant with that solution principle. Those solution principles in turn contribute to the achievement of subgoals, which are merged to the original top-level quality goals specified by the user (e. g. security, safety, usability).

Even though our approach combines the use of keywords and classification, it emphasizes the basic recommendation to prefer search via index [22].

The repository provides solutions, experiences and knowledge. It should be available publicly, for example as Public Domain. Nevertheless, there might be a need for managing private elements, for example for project-specific solu-

[1]Please note that for the sake of clarity only a subset of all dependency relationships are plotted.

tion within enterprises. Therefore the public repository can be extended by a project-specific one. Even if the project-specific extension is not visible outside, it is attached to the public repository seamlessly, i. e. there are dependency relations from elements of the project-specific part towards elements of the public one. Within the project-specific part, references to design issues within a project documentation can be drawn, representing architectural design tasks. These issues are linked to design alternatives, thus providing potential solutions as part of the documentation, determined by a tool and selected by a developer.

3.3 Description of Repository Elements

Architectural design decisions are driven by multiple criteria. Those criteria could be quality goals, organisational or social aspects (e. g. policies or skills), or previously chosen solutions that influenced the existing system causing constraints for subsequent design decisions. Those constraints may further limit the design space resulting from specified quality goals.

To take those constraints into consideration, solutions do not only have a textually or graphically modelled description, but these descriptions are enhanced by solution specific properties such as a library's version. Besides solution specific properties and beneficial consequences, a solution's description also includes adverse effects. Those adverse effects could be a negative impact on a quality goal specified or demands that are incompatible with existing solutions (e. g. wrong programming language). The dependency relationships are represented in the GSS by weighted edges: a positive value with a maximum of 1.0 means that a solution principle or quality goal benefits from a solution. Correspondingly, a negative value (min. -1.0) indicates a negative effect. These dependency relationships can not only be established across layers but also within the same layer, e. g. there could be a connection between RBAC and multitenancy in Figure 1.

Each solution can be rated and commented by users to share their experiences and their know-how with others. Recommendations are accessible through a selected solution's details. The rating should particularly afford to assess a solutions effect to a specific quality goal or solution principle. The comment could describe known implementation issues or give similar information about the experience made when applying the solution.

Yet, there are some adoptions necessary when representing security knowledge: When representing architectural knowledge that primarily is related to other quality attributes, it may be sufficient to provide patterns but to omit anti-patterns as a separate solution type. In the case of security, these anti-patterns are essential to prevent weaknesses and vulnerabilities. The same applies for known implementation issues that cause vulnerabiliets. These implementation issues may include information about common attacks on a solution.

4. POPULATING THE REPOSITORY WITH SOLUTIONS

4.1 Solving the Start-up Problem

When initially starting a solutions repository a start-up problem arises: to encourage users, a critical mass of solutions has to be provided by a repository, and to stimulate sharing of security knowledge, we need to create incentives for architects. I. e. they first have to benefit from a collaborative knowledge sharing approach before they are willing to share their own knowledge. Yet, it is usually impractical for researchers to populate solutions themselves beyond giving some examples. To approach that issue, we developed an approach to capture architectural knowledge from developer communities such as StackOverflow [24]. Prospectively, this approach can be applied to other sources of architectural knowledge including pattern catalogues and best practices.

4.2 Assessment and Maintenance of Solutions within the Repository

For providing valuable and correct recommendations, the repository's elements have to be kept up-to-date and comprehensive. For further enhancing the architectural decision support, we use diverse mechanisms:

Firstly, the approach described in Sect. 4.1 can not only be used for initially populating architectural knowledge, but information from blogs and forums can be used to periodically update knowledge represented within the repository. This especially concerns security solutions as there is an even more rapid technical progress. In this case, developer communities are a particularly valuable source of knowledge.

Secondly, we plan to connect the repository to existing services that for example publish security alerts or bulletins in order to keep information up-to-date. Concrete examples are Microsoft's monthly Security Bulletins[2] and the Android Security Bulletins[3].

As a third mechanism, the recommender system allows users to update a solution's impact on quality goals indirectly. In the same way users can connect quality goals to solution principles or solution principles to solutions instruments such as design principles. I. e. they can describe dependency relationships that were not represented so far. To do so, solutions can be rated in multiple categories that conform to the solutions description scheme and quality goals. Each category can be rated on a five-tier scale of -2 to +2: 0 means that there is no dependency relationship. A negative number means a minor or major negative impact, whereas a positive number means that the solution has positive impact on the referenced aspect. To gain a better assessment of a user's rating, significant contextual information is gathered, i. e. information on the specific project environment including important quality goals.

Besides browsing and rating them, users can also modify, add and delete solutions including their descriptions.

5. ENHANCING THE DEVELOPMENT PROCESS WITH THE REPOSITORY

The repository approach was mainly developed to support software architects and developers during the architecture design and evolution phase. It supports the implementation phase to some extend, e. g. through code-level design patterns, code templates, or tools.

Subsequently, we will concentrate on how our approach supports a software architect in making design decisions including trade-offs. Using our case study introduced at the beginning of this paper, an architect searches for design alternatives for multi tenancy.

[2]technet.microsoft.com/en-us/security/bulletins.aspx
[3]source.android.com/security/bulletin/

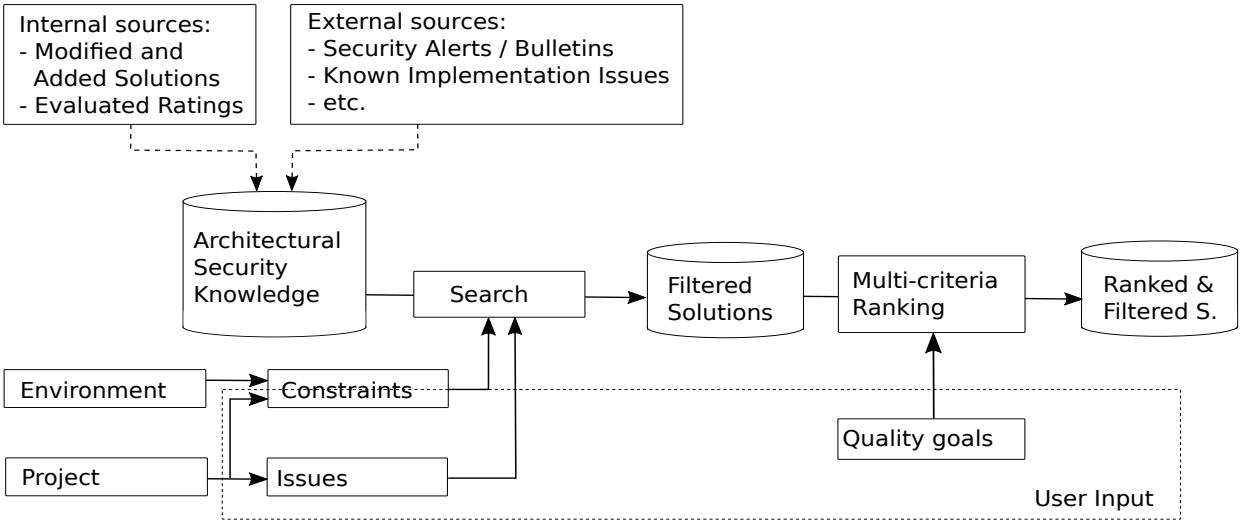

Figure 2: Architectural decision support through filtering and ranking solutions

In the first step, the architect describes the design issue he wants to solve as a search input. In our case, the architect creates two design issues: (1) How to design data isolation in multi tenant applications and (2) how to enable feature customisation in the same context.

There are several constraints that should be taken into consideration when making design decisions: those constraints are driven by environmental factors such as organisational policies or skills as well as by a possibly existing system, i. e., solutions that influenced its design. We differentiate between soft and hard constraints. Soft constraints could be suspended by the architect whereas hard constraints must be obeyed. In [11] we describe how those constraints can be identified in general.

Constraints are represented through so called Technical Terms [14]. Technical Terms consist of a meaningful identifier, a data type, and optionally a unit, e. g. in our case study that could be "Programming Language" as an identifier, "Java 1.7" (data type stringInt) and "ORM Mapping" with "Hibernate 5.2" as a value. In this case, the string "Java" would be further specified by the version "1.7" (value) combined with a relational operator such as equals or greater than.

There are two sources for constraints: they can be specified by the architect, or they originate from the solutions that have already been selected within the repository, i. e. solved design issues.

As Figure 2 shows, the search for candidate solutions can now be started with the architectural knowledge contained within the repository and the inputs specified before: issues and constraints. In order to perform this search, we combine rule-based and stochastic search strategies. The result is a set of filtered solutions. Next, this set has to be ranked by the quality goals specified by the user.

In our example case of issue (1) the search could for example result in the elements "separate database", "shared database with a separate schema", or "shared database and schema": Using separate databases for each tenant reduces the effort necessary for conducting changes according to the tenant's needs. In a shared database approach one database is provided for all tenants with each having a separate schema, i. e. an isolated set of tables. A third approach uses one shared database and a shared schema for all tenants. A tenant identifier column is used to associate each tenant with its records. Considering the proposed solutions, the separate databases approach is most secure, but has high maintenance effort, while the design alternative mentioned last is most vulnerable. Yet, a low modification effort is still an important quality goal for the case study's software system. A trade-off decision taking various criteria into consideration is needed. The repository supports the decision by ranking the previously filtered solutions: Over all the system recommends using a shared database with separate schema as it is ranked best. Our tool DecisionBuddy provides several ways of calculating a ranking, for example Arithmetic Mean and Pareto Optimum [11].

The filtered and ranked design alternatives are assigned to the initial design issue. To solve the design issue, the user can reject alternatives or select one.

Subsequently, the architect can share his experiences with the selected knowledge through the recommender system.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a repository-based approach for better reuse of security mechanisms and knowledge in software engineering. In addition to repositories in other architecture design approaches, we extend the information about abstract solutions and building blocks by engineering knowledge regarding security design, by information on relations and constraints regarding other solutions and system environment, by information on solutions' impact on quality properties, and by experiences from other developers. Furthermore, the repository supports project-tailored design decisions through a user-guided search regarding quality goal priorities, and regarding a solution's technological demands and constraints. A novelty of our approach in this context is

the combination of a classification through an ontology with a search-based approach exploiting explicit descriptions.

Our work focuses on security. Therefore we elaborated security-specific demands for the solution's representation including a particular need for anti-pattern or the consideration of security alerts and bulletins and updates on a regular basis.

The utilization of user recommendations for the assessment of solutions regarding their contribution to a solution principle or quality goal allows us to collect the experiences of multiple users and across projects.

As a part of our future work we want to facilitate the maintenance of repositories, especially regarding frequent updates of security information and on technology features. Accordingly, we work on a concept for coupling our repository with existing sources of information by knowledge capturing techniques. Such sources of information can for example be security alerts and bulletins.

As another important part of our future work we started to conduct qualitative interviews to investigate the current state of practice regarding the use of different types of knowledge about proven security solutions during system design. Therefor, we plan to interview software engineers with regard to the documentation of a system's security aspects and their consideration during the software architecture design.

In order to better present security solutions and to facilitate the explicit consideration of security concepts during architecture design, we perform further research on an integrated modelling concept for security and other architectural properties. For this purpose it will first be necessary to answer the question, whether and in what manner a model that abstracts from details could be adequate for achieving security. I. e., we plan to focus on determining the appropriate level of abstraction first, to ensure the model's capability for security design as well as comprehensibility and maintainability of the models. Those models can then be used for considering security constructively during software system design as well as for a more comprehensible description of solutions within the repository.

Besides the design phase, we plan to work on better support for the implementation phase. Therefore, we want develop a concept for software developer assistance by marking code blocks as relevant for security. In this way we want to increase other developer's awareness when modifying the code and, hence, prevent them from introducing weaknesses. Markings of code blocks for developers shall be enhanced by automatically generated warnings. These warnings would result from an analysis that attempts to identify anti-patterns contained in our repository. The latter could also be a valuable assistance in security code reviews.

7. ACKNOWLEDGMENTS

We would like to thank all anonymous reviewers for their valuable hints and suggestions to further improve our approach. All comments will be considered in our future work.

8. REFERENCES

[1] M. A. Babar and I. Gorton. A Tool for Managing Software Architecture Knowledge. In *SHARK/ADI '07: ICSE Workshops 2007. 2nd Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent*, pages 11–11, May 2007.

[2] D. Basin, J. Doser, and T. Lodderstedt. Model Driven Security: From UML Models to Access Control Infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15(1):39–91, 2006.

[3] A. v. d. Berghe, R. Scandariato, K. Yskout, and W. Joosen. Design Notations for Secure Software: A Systematic Literature Review. *Software & Systems Modeling*, 2015.

[4] S. Bode. *Quality Goal Oriented Architectural Design and Traceability for Evolvable Software Systems*. PhD thesis, Ilmenau University of Technology, Ilmenau, Germany, April 2011.

[5] S. Bode and M. Riebisch. *Tracing the Implementation of Non-Functional Requirements*, pages 1–23. IGI Global, 2011.

[6] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas. A Web-based Tool for Managing Architectural Design Decisions. *SIGSOFT Software Engineering Notes*, 31(5), Sept. 2006.

[7] D. Falessi, G. Cantone, R. Kazman, and P. Kruchten. Decision-making Techniques for Software Architecture Design: A Comparative Survey. *ACM Computing Surveys*, 43(4):33:1–33:28, Oct. 2011.

[8] R. Farenhorst, R. Izaks, P. Lago, and H. v. Vliet. A Just-in-time Architectural Knowledge Sharing Portal. In *WICSA 2008. 7th Working IEEE/IFIP Conference on Software Architecture*, pages 125–134, Feb 2008.

[9] E. B. Fernandez, P. Cholmondeley, and O. Zimmermann. Extending a Secure System Development Methodology to SOA. In A. M. Tjoa and R. R. Wagner, editors, *18th International Workshop on Database and Expert Systems Applications*, pages 749–754, 2007.

[10] S. Gerdes, S. Lehnert, and M. Riebisch. Combining Architectural Design Decisions and Legacy System Evolution. In P. Avgeriou and U. Zdun, editors, *Proceedings of the 8th European Conference on Software Architecture: ECSA 2014*, pages 50–57, Cham, 2014. Springer International Publishing.

[11] S. Gerdes, M. Soliman, and M. Riebisch. Decision Buddy: Tool Support for Constraint-based Design Decisions During System Evolution. In *1st International Workshop on Future of Software Architecture Design Assistants (FoSADA)*, pages 1–6, May 2015.

[12] J. Jürjens. *Principles for Secure Systems Design*. Dissertation, Oxford University, University of Oxford, 2002.

[13] J. Jürjens. Foundations for Designing Secure Architectures. *Electronic Notes in Theoretical Computer Science*, 142:31–46, 2006.

[14] A. Pacholik and M. Riebisch. Modelling Technical Constraints and Preconditions for Alternative Design Desicions. In *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VIII*, pages 101–106, 2012.

[15] J. Ren and R. N. Taylor. A Secure Software Architecture Description Language. In *Proceedings of the Workshop on Software Security Assurance Tools, Techniques, and Metrics*, 2005.

[16] J. Ren, R. N. Taylor, P. Dourish, and D. F. Redmiles. Towards an Architectural Treatment of Software

Security: A Connector-centric Approach. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.

[17] M. Riebisch, S. Bode, and R. Brcina. Problem-solution Mapping for Forward and Reengineering on Architectural Level. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, pages 106–115, New York, NY, USA, 2011. ACM.

[18] T. L. Saaty. Decision Making with the Analytic Hierarchy Process. *International Journal of Services Sciences*, 1(1):83, 2008.

[19] M. Schumacher. *Security Engineering with Patterns: Origins, Theoretical Model, and New Applications*, volume 2754 of *Lecture Notes in Computer Science*. Springer, Berlin and Heidelberg, 2003.

[20] M. Schumacher. *Security Patterns: Integrating Security and Systems Engineering*. Wiley series in software design patterns. John Wiley & Sons, Chichester, England and Hoboken, NJ, 2006.

[21] M. Shahin, P. Liang, and M. R. Khayyambashi. Architectural Design Decision: Existing Models and Tools. In *WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on Software Architecture, 2009 European Conference on Software Architecture*, pages 293–296, Sept 2009.

[22] M. Shaw. Keynote: Progress Toward an Engineering Discipline of Software. In *Presentations from the program of SATURN 2015 (April 27-30, 2015, in Baltimore, Maryland)*. Software Engineering Institute, Software Engineering Institute, 2015.

[23] L. Sion, K. Yskout, A. v. d. Berghe, R. Scandariato, and W. Joosen. Masc: Modelling Architectural Security Concerns. In *IEEE/ACM 7th International Workshop on Modeling in Software Engineering (MiSE)*, pages 36–41, 2015.

[24] M. Soliman, M. Galster, A. R. Salama, and M. Riebisch. Architectural Knowledge for Technology Decisions in Developer Communities. In *13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2016.

[25] M. Soliman and M. Riebisch. Modeling the Interactions between Decisions within Software Architecture Knowledge. In *Proceedings of the 8th European Conference on Software Architecture: ECSA 2014*, pages 33–40, 2014.

[26] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar. A Comparative Study of Architecture Knowledge Management Tools. *Journal of Systems and Software*, 83(3):352–370, Mar. 2010.

[27] E. Triantaphyllou. *Multi-criteria Decision Making Methods: A Comparative Study*, volume 44 of *Applied Optimization*. Springer, Boston, MA, 2000.

[28] S. Vijayalakshmi, G. Zayaraz, and V. Vijayalakshmi. Article: Multicriteria Decision Analysis Method for Evaluation of Software Architectures. *International Journal of Computer Applications*, 1(25):22–27, February 2010. Published By Foundation of Computer Science.

[29] B. Wynar and A. Taylor. Introduction to Cataloging and Classification. Libraries Unlimited. *Inc.*, 1992.