Evaluation of Engineering Approaches in the Secure Software Development Life Cycle*

Marianne Busch, Nora Koch, and Martin Wirsing

Institute for Informatics
Ludwig-Maximilians-Universität München
Oettingenstraße 67, 80538 München, Germany
{busch,kochn,wirsing}@pst.ifi.lmu.de

Abstract. Software engineers need to find effective methods, appropriate notations and tools that support the development of secure applications along the different phases of the Software Development Life Cycle (SDLC). Our evaluation approach, called SecEval, supports the search and comparison of these artifacts. SecEval comprises: (1) a workflow that defines the evaluation process, which can be easily customized and extended; (2) a security context model describing security features, methods, notations and tools; (3) a data collection model, which records how data is gathered when researchers or practitioners are looking for artifacts that solve a specific problem; (4) a data analysis model specifying how analysis, using previously collected data, is performed; and (5) the possibility to easily extend the models, which is exemplarily shown for risk rating and experimental approaches. The validation of SecEval was performed for tools in the web testing domain.

1 Introduction

The development of software requires among others decisions regarding methods, notations and tools to be used in the different phases of the Software Development Life Cycle (SDLC). In the development of secure applications, such decisions might even be more relevant as new threats continuously appear and more and more methods, notations and tools are developed to increase the level of security. Therefore it is important to be able to identify, e.g., authentication-related threats that can be mitigated by a method and to find out which tools support this method. Furthermore, it is advantageous to know which tools can work together.

However, often the selection of methods, tools and notations is performed based on the experience of the developers, as all too frequent there is neither time to investigate on alternatives to the artifacts used so far, nor to document choices and lessons learned. In other cases engineers have to search in a time-consuming process for appropriate artifacts, decide about the relevant research

* This work has been supported by the EU-NoE project NESSoS, GA 256980.

questions and repeat evaluations. What could help is a systematic way of collecting information on methods, tools and notations driven by specific research questions and a subsequent selection.

To ease the tasks of recording information and of getting an overview of existing artifacts the Common Body of Knowledge (CBK) [1] was implemented as a semantic Wiki within the scope of the EU project NESSoS [2]. It provides a useful knowledge base and underlying model, but leaves open the following questions: (a) How could security-related features also be included as first-class citizens in the knowledge base? (b) How can we use the approach not only for recording and comparing features of methods, notations and tools, but also for documenting the search process? (c) How is the process of data collection and data analysis specified, to make sure that emerging research results are comprehensible and valid?

The aim of our evaluation approach is to give answers to these questions and to provide software and security engineers with mechanisms to ease the selection and comparison of methods, tools and notations. Our conceptual framework for evaluating security-related artifacts is called SecEval [3,4]. We selected a graphical representation for SecEval, which comprises (1) a workflow that defines the evaluation process, which can be easily customized and extended; (2) a security context model describing security properties, vulnerabilities and threats as well as methods, notations and tools; (3) a data collection model, which records how data is gathered when researchers or practitioners do research to answer a question; and (4) a data analysis model specifying, how reasoning on the previously collected data, is done. However, we do not claim to provide a one-fits-all approach for IT-security (which would overload any model), instead we introduce an extensible basis.

In this chapter we focus on the evaluation process and its placement within the software development life cycle. Conversely to [3] in which we presented the architectural features of SecEval, we go into more details, concerning its requirements, the process supported by SecEval, and the case study. In addition, we show how the conceptual framework can be extended to cover approaches like the OWASP's Risk Rating Methodology [5] and Moody's method evaluation approach [6]. The applicability of an approach like SecEval is given by an appropriate tool support and the usability of its user interface. Therefore we plan an implementation of SecEval as an online knowledge base and present the requirements of such an implementation.

The remainder of this chapter is structured as follows: Section 2 discuss related work and background. Section 3 gives an overview of the SDLC for the development of secure software and systems. Section 4 describes our evaluation approach SecEval in detail, before Sect. 5 presents its extensions. In Sect. 6 we validate the approach by a case study in the area of security testing of web applications. We give an overview of the requirements for a future implementation of SecEval in Sect. 7 and conclude in Sect. 8.

2 Related Work

In this section, we discuss approaches that focus on security during the Software Development Life Cycle (SDLC). We continue with general evaluation approaches and conclude with security-specific evaluation frameworks.

Secure Software Development Life Cycles. Incorporating security into the SDLC means to add activities to ensure security features in every phase of SDLC. Adopting a secure SDLC in an organization's framework has many benefits and helps to produce a secure product. These approaches enrich the software development process by, e.g., security requirements, risk assessment, threat models during software design, best practices for coding, the use of static analysis code-scanning tools during implementation, and the realization of code reviews and security testing. Hereby, the concrete phases of the SDLC and how they are arranged is less important than the focus on security during all phases.

Therefore, many companies define their own secure SDLC in order to be able to ensure the software they developed has as few vulnerabilities as possible. A main contribution in this area is the Microsoft Security Development Lifecycle (SDL) [7]. It is a software development process used to reduce software maintenance costs and increase reliability of software concerning software security-related bugs. The SDL can be adapted to be used in a classical waterfall model, a spiral model, or an agile model.

Besides, cybersecurity standards, as ISO 27001 [8] can be applied. They go beyond software development and define an information security management system that requires the specification of security guidelines for policies, processes and systems within an organization. Important is that most standards do not define how to increase security, but which areas have to be taken into consideration in order to create meaningful security guidelines.

Another example for supporting secure development along the SDLC is the Open Web Application Security Project (OWASP). It comprises, beyond others, a set of guides for web security requirements, cheat sheets, a development guide, a code review and a testing guide, an application security verification standard (ASVS), a risk rating methodology, tools and a top 10 of web security vulnerabilities [9].

General Evaluation Approaches. Our approach is based on the so called "Systematic Literature Review" of Kitchenham et al. [10], which is an evaluation approach used in software engineering. Their aim is to answer research questions by systematically searching and extracting knowledge of existing literature. We go even further using arbitrary resources in addition to available literature, such as source code or experiments that are carried out to answer a research question. The systematic literature review is executed in three main steps: first, the review is planned, then it is conducted and finally results are reported (this process is depicted in deliverable D5.2 [11, Fig. 3.2]). In contrast to Kitchenham's approach, our data collection process is iterative, and more specific for a chosen

domain as we specify a detailed structure of the context for which we pose the research questions.

The CBK (Common Body of Knowledge) [11] defines a model to collect and describe methods, techniques, notations, tools and standards. We use the CBK as a starting point for our SecEval's approach. However, in our model we do not consider standards and we aggregate the concepts of technique and method, as an instance model immediately shows whether actions (in our case called steps) are defined. In contrast to the CBK, SecEval focuses on security-related features providing a fine-grained model. In addition, it defines a process for the evaluation of methods, tools and notations. The CBK is implemented as a semantic Wiki [1] and serves as a knowledge base to which queries can be posted. Unlike the CBK, SecEval is not implemented yet.

SIQinU (Strategy for understanding and Improving Quality in Use) [12] is an approach defined to evaluate the quality of a product version. It is based on the conceptual framework C-INCAMI (Contextual-Information Need, Concept model, Attribute, Metric and Indicator), which specifies general concepts and relationships for measurement and evaluation. The latter consists of six modules: measurement and evaluation project definition, nonfunctional requirements specification, context specification, measurement design and implementation, evaluation design and implementation, and analysis and recommendation specification. Although C-INCAMI is used for the domain of quality evaluation, we recognized several properties we considered relevant for our approach. Regarding the framework specification technique SIQinU provides an evaluation strategy that is sketched as UML activity diagrams whereas C-INCAMI concepts and relationships are specified as a UML class diagram. We also stick to use UML for graphical representation of our approach and implemented separation of concerns through UML packages.

Moody [6] proposes an evaluation approach which is based on experiments and centers the attention on practitioners' acceptance of a method, i.e. its pragmatic success, which is defined as "the efficiency and effectiveness with which a method achieves its objectives". Efficiency is related to the effort required to complete a task and effectiveness to the quality of the result. In [6] practitioners use methods and afterwards answer questions about perceived ease of use, perceived usefulness and intention to use. This approach is integrated into SecEval (cf. Sect. 5).

Security-specific Evaluation Approaches. Security-related frameworks often consider concrete software systems for their evaluation. An example is the OWASP Risk Rating Methodology [5], where the risk for a concrete application or system is estimated. We added vulnerability-dependent features of the OWASP model to SecEval, as e.g., the difficulty of detecting or exploiting a vulnerability. Features that are related to a concrete system and the rating of a possible attack are introduced as an extension of SecEval, which can be found in Sect. 5.

The i* [13] metamodel is the basis of a vulnerability-centric requirements engineering framework introduced in [14]. The extended, vulnerability-centric i* metamodel aims at analyzing security attacks, countermeasures, and re-

quirements based on vulnerabilities. The metamodel is represented using UML class models.

Another approach that focuses on vulnerabilities is described by Wang et al. [15]. Their concept model is less detailed than the i* metamodel. They create a knowledge base that can be queried using the Semantic Web Rule Language (SWRL) [16]. Unlike our approach, they do not use graphical models.

Moyano et al. [17] provide a Conceptual Framework for Trust Models which is also represented using UML. As trust is an abstract concept, which emerges between communication partners, we do not consider it in SecEval.

3 Engineering Secure Software and Systems

In the NESSoS project, we address the development of secure software and systems starting from the early phases of the secure Software Development Life Cycle (SDLC). The life cycle on which this section is based [18], considers not only the traditional phases, such as requirements engineering, design, implementation, testing and deployment, but also stresses the relevance of service composition and adaptation. In addition, we have to ensure that the developed software is secure; therefore we include assurance as an orthogonal topic of paramount importance. Another aspect is risk and cost awareness, which is a key research direction we foresee also as transversal since it links security concerns with business.

Figure 1 gives an overview of tools, methods and notations for which information is available in the NESSoS Common Body of Knowledge (CBK) [1] relating them to the phases of the SDLC in which they can be used. The graphical representation includes both the traditional phases and the orthogonal ones mentioned above. On the left bottom corner Fig. 1 includes tools and methods that correspond to a metalevel as they help developers to build more secure software along the whole SDLC: On the one hand the Service Development Environment (SDE) and the CBK, which were already implemented and are available online. On the other hand two methods: our evaluation framework for security artifacts (SecEval) and the Microsoft Security Development Lifecycle (SDL) [7].

The CBK provides the descriptions of methods, tools, notations and standards in the area of engineering secure software; several of the tools described in the CBK are integrated in the SDE tool workbench, which allows for connecting and executing tools in a toolchain (see chapter [19]). The amount of these artifacts that support the different phases makes it difficult to select the appropriate ones for a project, lead to the development of the SecEval approach that provides a systematic evaluation and comparison of methods, notations and tools, which is further detailed in Sect. 4.

Security Requirements Engineering. The main focus of the requirements engineering phase is to enable the modeling of high-level security requirements of the system under construction. These requirements can be expressed in terms of concepts such as compliance, privacy or trust and should be subsequently mapped

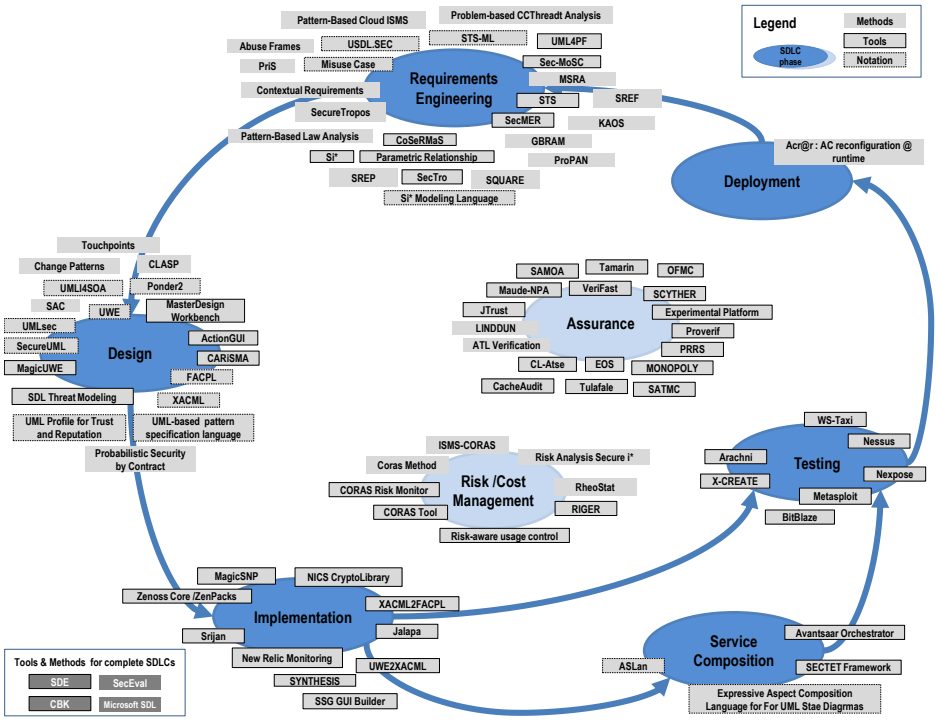

Fig. 1. Overview of Security-Related methods, notations and tools in the SDLC

into more specific requirements that refer to technical solutions. Indeed, it is important that security requirements are addressed from a higher-level perspective, e.g., in terms of the actors' relationships with each other and by considering security not only at the technological level. It is essential to analyze how security may impact on other functional and non-functional requirements, including Quality of Service/Protection (QoS/P), both at design-time and at run-time. In this respect, agent-oriented and goal-oriented approaches such as Secure Tropos [20] and KAOS [21] are currently well recognized as means to explicitly take the stakeholders' perspective into account.

Elicitation, specification – in particular modeling and documentation – and verification of security requirements is a major challenge and will be even more relevant in applications in the Future Internet (FI), as systems are becoming more autonomous [22]. A significant number of methods has been proposed to elicit and analyze security requirements, but there is a lack of empirical comparative evaluations that support decisions to favor one over another. The SecEval approach was developed to close this gap.

For the evaluation it is important to define the relevant research objectives, i.e. which are the criteria for a comparison and selection. For example: "Which methods exist that support the elicitation of security requirements for the embed-

ded domain?" or "Which notations can be used for the specification of security requirements of web applications?"

Design of Secure Software and Services. Separation of concerns is an essential principle of software design that aims at managing the growing complexity of software intensive systems [23]. Since the early 2000's this software engineering principle has been integrated in model-driven engineering, through a large research and tooling effort. The aim is to provide convenient modeling techniques that enable developers not only to graphically represent what customers need regarding security in a concrete and intuitive manner, but also to seamlessly implement it afterwards in any selected framework.

For example, modeling access control is currently supported in two major UML-based methods: UMLsec [24] and SecureUML [25]. For web applications the methods ActionGUI [26] (cf. chapter [27]) and UWE [28] (cf. chapter [29]) also extend UML. Both methods model access control using a variant of SecureUML. Additionally, concepts as authentication and privacy, besides others, are taken into account.

Alternative domain-specific languages emerged, which allow stakeholders with heterogeneous backgrounds to model their concerns in the early development phases. This reduces the cognitive distance between the abstract formal concepts and domain experts' knowledge, reduces the risk of errors in requirements elicitation and can thus drastically improve the quality of the implemented system. As an example, recent work integrated security concerns in a business modeling language to let project managers and company executives reason on security issues on models expressed in concepts they can apprehend [30]. Other works include access control policy enforcement mechanisms generated automatically from high-level requirements models. The policies need to be submitted to checks in order to ensure security aspects being modeled are preserved in the code [31].

Effective methods and tools to deal with security concerns in design models are needed to manage the major threat of increasing cost to deploy, fix and maintain security mechanisms. If we are able to design abstract models for these concerns, they are much more difficult to understand at the code level, and even more difficult to maintain, because of all the technical details introduced at the code level.

The selection of appropriate methods and corresponding tools for the design of secure applications remains a crucial decision and definitely will influence other phases of the SDLC. A typical research question could be: "Are there any tools that support secure web engineering and that can be used by non-experts?" or "Which UML CASE tools support model-driven development with reduced learning effort?"

Implementation of Secure Applications. Many security vulnerabilities arise from programming errors that allow for their exploitation. For example, the OWASP top ten list [9] for web application security flaws, clearly shows how coding issues as injection, cross scripting and generally speaking wrong programming practices are major issues to be tackled. The aim is therefore to use languages and tools

that minimize this threat. This can be partially achieved by emphasizing the use of well-known programming principles and best practices in secure software development. In particular, language extensions or security patterns can be used during development to guarantee adherence to best practices.

The main focus of this research area is not only language based security, secure coding principles and practices but also programming platforms enforcing security properties. Indeed, reliable programming environments are crucial to minimize the presence of exploitable vulnerabilities in software-based services. Research questions for selecting appropriate languages, methods and programming environments are: "What are common security flaws in applications implemented with C++?" and "Which methods and tools exists to harden the application against these vulnerabilities?"

Testing. The implemented software has to be tested in different ways; both if it fulfills the structural and functional goals, i.e. it has to be checked whether the requirements are all achieved and it has to be tested for bugs. In particular, security testing consists of verifying that data is safe and of ensuring functionality.

The following are typical research questions that could be defined for the selection of appropriate vulnerability scanners for web applications. "Which vulnerability scanners are available for testing security features of web applications?", "Can these scanners run on Microsoft Windows, be freeware or provide at least a free trial version and come with a command line or web interface?"

Deployment. When deploying applications at the end of the build process, this is the appropriate moment to evaluate runtime characteristics of the applications in the context of the real environment. Deployment reviews for security focus on evaluating security design and implementation and the configuration of the application and the network. The objective is to verify if the settings of the web server are correct like the configuration of file encryption, the use of authentication and the applied personalization issues. Within the scope of the Microsoft SDL, a checklist for the deployment reviews is provided, which includes, e.g., checks for latest patches, installed updates or strong passwords. A research question that arises in the deployment context is "Which methods support systematic deployment reviews?"

Service Composition and Adaptation. The capability to achieve trustworthy secure composition is a main requirement in security engineering. Building secure services that cannot be further composed is an inherent obstacle that needs to be avoided. The integration and interoperability of services in order to tailor and enhance new services require adapting the service interfaces at different levels, including the semantic level. Another aspect to consider include assessing the trustworthiness of composition of services.

Integration and interoperability of services, is achieved among others using techniques such as semantic annotations and secure adaptation contracts, as well as decentralized secure composition and distributed component models. Services and components need to be more open, with clearer interfaces and need to be

easily accessible from known repositories. Moreover, a research question could investigate for example techniques that provide security measures for composed services [18].

Assurance. During the SDLC, there is a need to ensure security from many perspectives. On the one hand, the security design decisions and the choices of security mechanisms that are used must fulfil the identified security requirements. On the other hand, it is important that engineers are able to select the appropriate mechanisms for implementing required security features.

As shown in Fig. 1 many tools were implemented to check different security aspects of software that is under development. The focus of the assurance activities are: (1) Security support for service composition languages; (2) Run time and platform support for security enforcement; and (3) Security support for programming languages, aiming for verification. For example, tools such as Dafny [32] (for Dafny programs) and Verifast [33] (for C and Java programs) address assurance aspects in order to verify correctness, i.e. that software fulfills their requirements. A research question regarding methods and tools could be: "Which are helpful tools for assessing the trustworthiness of a system under development?"

Risk and Cost Management. The value of security solutions and their return on investment must be demonstrated from a business oriented perspective. Therefore, risk analysis plays an important role when selecting security solutions and implementing security measures. The integration of risk and cost analysis in the whole SDLC, and an extension of the overall approach towards execution time, is the necessary response to these needs.

The main objective of the identification and assessment of risks and the analysis of the costs of associated countermeasures is to exploit an engineering process that optimizes value-for-money in terms of minimizing effective risk, while keeping costs low and justified. A set of methods and tools are available in this context, among others those of the CORAS tool suite [34].

Relevant research questions in the area of security risk and associated cost management are: "What are most appropriate methodologies for performing risk management and cost assessment through the complete SDLC?" and "Which tools support conduction of risk management?"

4 Systematic Evaluation of Engineering Approaches

This section, which is an extension of [3], provides the description of SecEval, a conceptual evaluation framework for methods, notations and tools supporting the development of secure software and systems. The framework can be used to collect security-related data and to describe security-relevant metrics, using them for reasoning and obtaining the appropriate techniques for a specific project. An example for a simple evaluation is required to answer the question posted in the implementation phase: "Which library for authentication should be used?"

A more elaborated one could be the evaluation of risks for a concrete software system, which is a question that is relevant for all SDLC phases.

The conceptual framework comprises a structural part and a behavioral part, defined as a model for evaluation and an evaluation process, respectively. For the graphical representation of the evaluation model a UML class diagram was chosen; the evaluation process is represented as a UML activity diagram. In the remainder of the section, we present the requirements engineering work done to elicit the main steps of the process, followed by the main concepts of SecEval.

4.1 Evaluation Process

We start by eliciting the requirements of such a framework, i.e. which stakeholders are involved, which concepts play a role in secure software and evaluation of methods, tools and notations, and how those concepts are related. Therefore, the first step was to name common stakeholders for secure software: security engineers (i.e. security designers and developers), normal users of the software and attackers. In some cases, users can attack software without being aware of it, e.g., when they have a virus installed on their computer. We consider those users also attackers, as well as developers which are, e.g., trying to smuggle malicious code into software. Figure 2 depicts stakeholders and use cases in a UML use case diagram.

We grouped use cases based on their purpose in evaluation and development use cases. The `Evaluation` package at the top contains all use cases related to collecting, reasoning and selecting, e.g., tools, whereas the package `Development` at the bottom of the diagram refers to security-related tasks within the SDLC, such as identification of security requirements, design and implementation of these security requirements, identification and patch of vulnerabilities. The ≪include≫ dependencies show the order these use cases have in the SDLC: implementing secure software requires having a design, and a design implies that requirements were elicited beforehand. Both, the attacker and the security engineer can identify vulnerabilities, whereas the former usually attacks them and the latter tries to patch them, which is modeled using an ≪extend≫ dependency. Those patches can then be installed by users (which also might happen by using an automatic update function).

From time to time, tasks within the development package require evaluation activities to respond for example to questions like “Which tool should be used for gathering security requirements or for designing secure web applications?”. In fact, for security experts it is helpful to be aware of common security methods and tools that can be used for a specific task. For further examples of research questions related to the different SDLC phases the reader is referred to Sect. 3.

Figure 3 depicts the process of working with SecEval, which is represented as a UML activity diagram. The first step of the process is the data collection based on the defined research questions. Therefore different sources (as papers, reports, websites, ...) are gathered, which are then analyzed in the second step. This analysis process consists of extracting information from the data collected, activating some reasoning activities and expressing the results using SecEval's

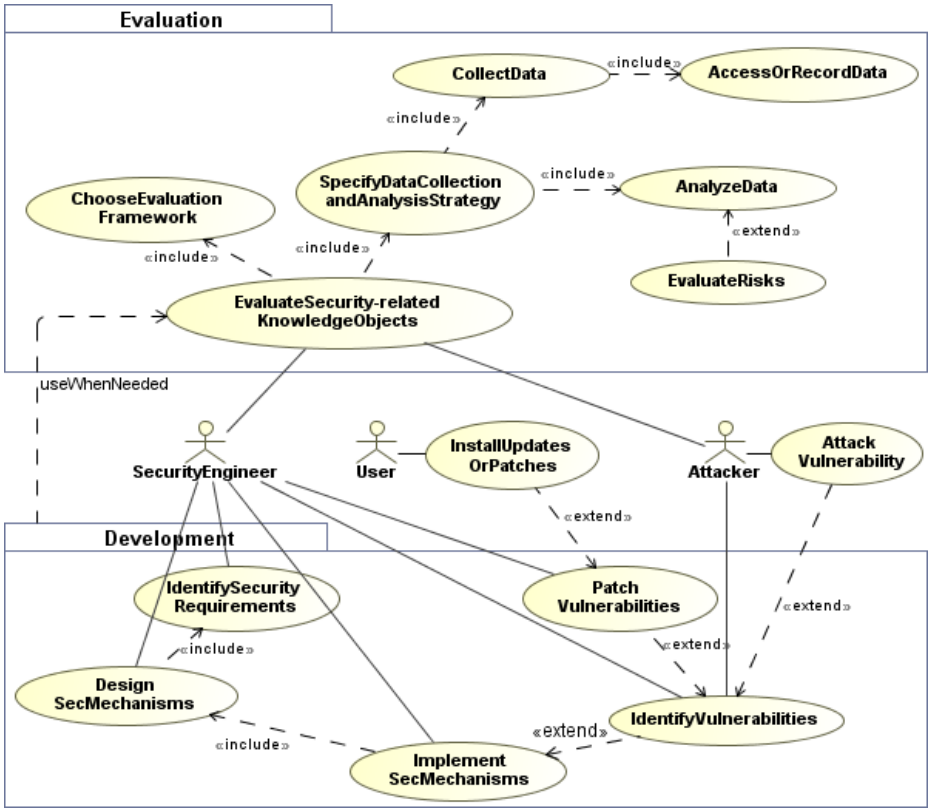

Fig. 2. Stakeholders and Use Cases

security context model. Notice that this process has to be adapted (and usually simplified) for a specific evaluation. Writing down the exact process might not always be necessary, as many tasks can be executed in parallel or in any order (indicated by horizontal bars).

In practice the basic ingredients of the evaluation process are a set of tasks that has to be performed and information pieces relevant for these tasks. Tasks are represented as UML activities like select queries, execute search/experiments and define filters. Information pieces are represented as objects in the UML model showing which input is required for a task and which are the results. Examples for identified objects are: research question, used resource, query, filter and criterion.

4.2 Systematic Evaluation – Model Overview

The use cases from our requirements analysis and the objects of the evaluation process were a starting point to identify relevant concepts related to security for

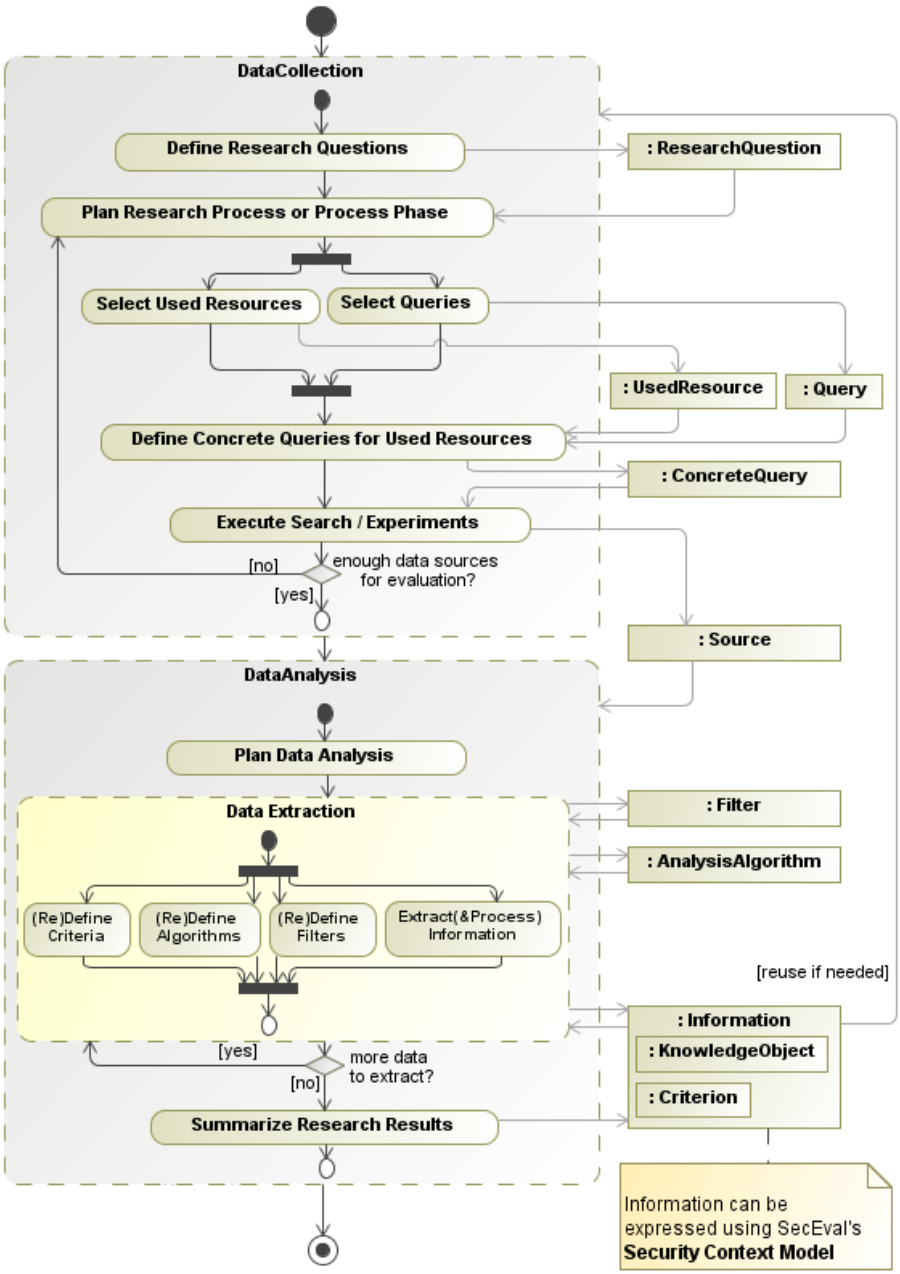

Fig. 3. SecEval's Evaluation Process

using and evaluating methods, notations and tools during the software engineering process. We clustered these concepts in three packages: Security Context, Data Collection and Data Analysis. Figure 4 shows the model represented as a UML class diagram that can be instantiated with concrete methods, tools and notations whenever needed.

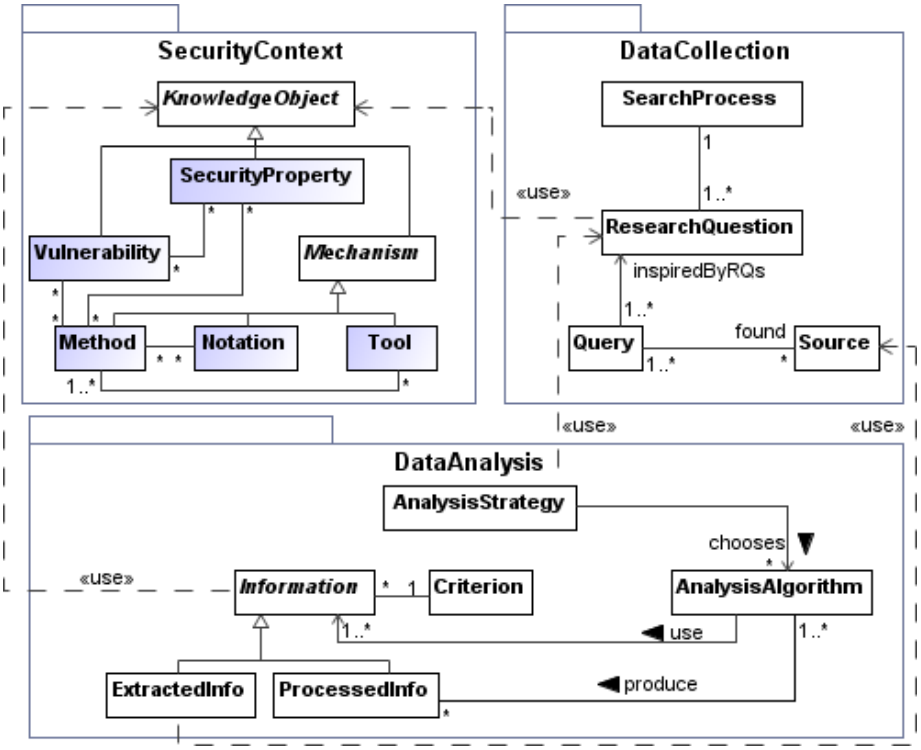

Fig. 4. SecEval: Model Overview [3]

4.3 Security Context

The Security Context package provides a structure for the classification of (security-related) methods, notations and tools together with security properties, vulnerabilities and threats. Within this package we represent a security feature as a class element and introduce an abstract class `Mechanism` from which the classes `Method`, `Notation` and `Tool` inherit common attributes such as `goals`, `costs`, `basedOnStandards`, etc. We focus on security aspects, but the model can also record non-security mechanisms.

In Fig. 5, for convenience enumerations' texts are grey and the background of classes which can directly be instantiated is colored. All attributes and roles are

typed; however the types are not shown in the figures due to brevity. The main characteristics of the class `Mechanism` are specified as boolean types (can.., has.., is..). In an implementation of our model, it should be possible to add further items to the enumerations.

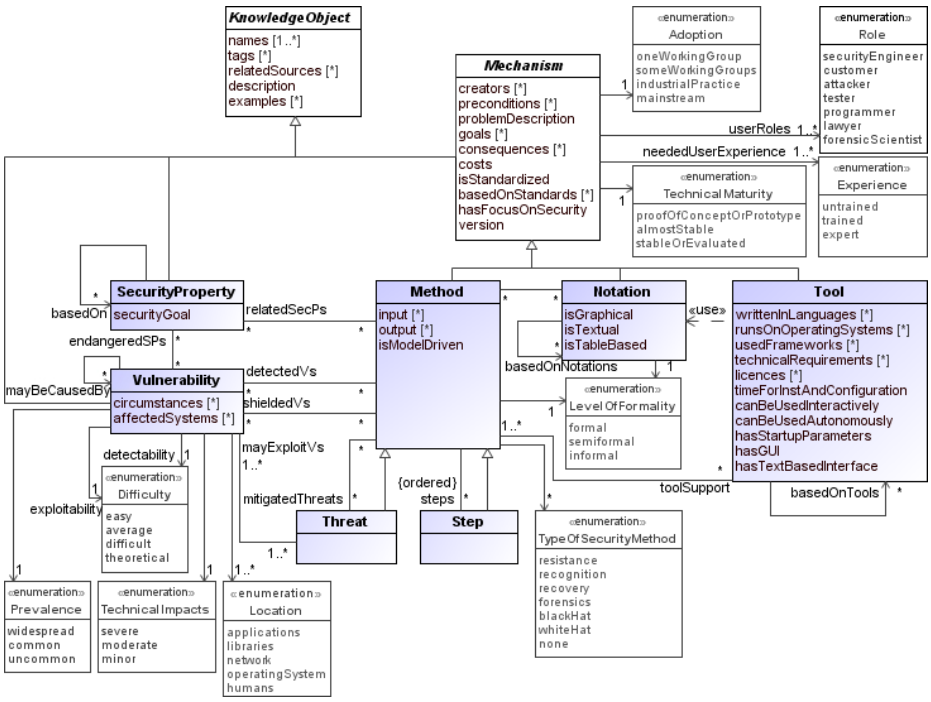

Fig. 5. SecEval: Security Context [3]

A Mechanism is described by a problem statement, by the goals it strives for, by its costs and by the consequences it implies. Mechanisms can be based on standards or be standardized themselves. Before applying a mechanism, the preconditions that are necessary for using it have to be fulfilled. Furthermore, an estimation regarding technical maturity and adoption in practice should be given. Several levels of usability can be stated indicating the experience users need in order to employ a mechanism, e.g., they need to be experts.

The classes Method, Tool and Notation inherit all these properties from the class `Mechanism` and have their own characteristics defined by a set of specific attributes. For example, a Method has some general attributes, such as input, output and if it is model-driven. These attributes are used to describe the method at a high level of abstraction. Note that a method or step can be supported by

notations or tools. These facts are represented in the model with corresponding associations between the classes.

For a NOTATION, we consider characteristics such as whether the notation is graphical, textual or based on a tabular representation. We also added a level of formality, which ranges from informal to formal. Notations can be based on other notations, for example many context-specific extensions for UML exist.

The description of a TOOL is given among others by the information of languages it is written in, operating systems it supports, frameworks it uses and licenses under which it is released. A tool can be based on other tools, which is the case when libraries are used or when plugins are written.

A distinguishing characteristic of our evaluation framework SECEVAL is the refinement of methods and tools based on the phases of the SDLC. As far as we know, no phase-related attributes are needed to describe features of notations.

Figure 6 depicts our `Tool` class and the abstract class `TAreasOfDev`, which is a wildcard for detailed information about the tool in relationship to the phases of the NESSoS SDLC [35]: requirements, design, implementation, testing, assurance, risk & cost management, service composition and deployment. We added an additional category to distinguish methods and tools that operate at the runtime of a system. A tool can eventually support several development phases. The meaning of attributes should be self-explaining, but is described in more detail in [36].

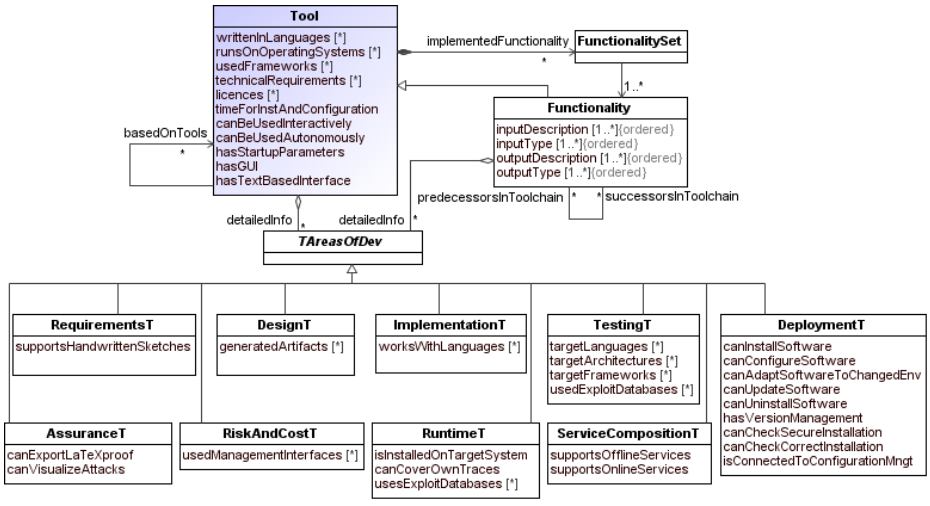

Fig. 6. Security Context: Details of Tools

Similarly, a method can be redefined according to the phases in the SDLC it covers, as depicted in Fig. 7. For example a method, such as Microsoft's

Security Development Lifecycle [7], can be used as a basis for designing secure applications, but also covers other phases. In this case, the attributes of the classes `DesignM` and `ImplementationM` and others would be used to describe this method.

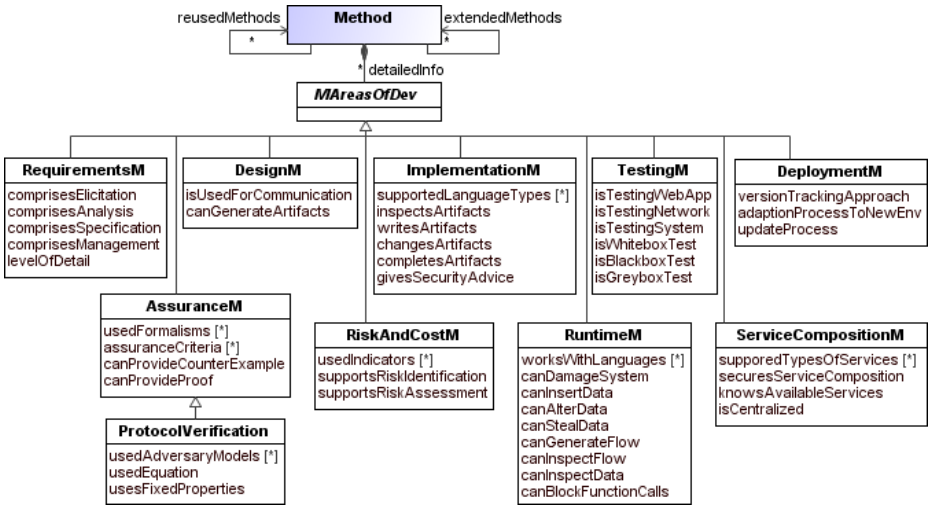

Fig. 7. Security Context: Details of Methods [3]

As seen before, a tool supports a certain method. However, we have not yet defined the quality of this support. Does the tool fully support the method? Does it provide partial support? Which features are not supported? We add this information to the model using the association class `ToolSupportedMethod`, as depicted in Fig. 8 with a dotted line. The association class itself is inherited from the class `Method`, thus can redefine its attributes. For instance, a design tool can partly support a model-driven method (e.g., by facilitating the modeling process), although it cannot generate artifacts. In this case, DesignM.canGenerateArtifacts (cf. Fig. 7) would be set to false.

A method can extend other methods, which means it might also change them. In this case the role `extendedMethods` should be further specified, we recommend to add an association class which inherits from the class `Method` (similar to the association between method and tool). In this way, it can be exactly described if and how the original methods are modified. It is also possible that other methods are used without any changes (role `usesMethods`).

We adopted the abstract class KnowledgeObject which is used in the CBK as a super class for all elements which are described by SecEval. In SecEval, we applied separation of concerns so that only very general descriptions remain as attributes in a knowledge object, which can be applied to all elements

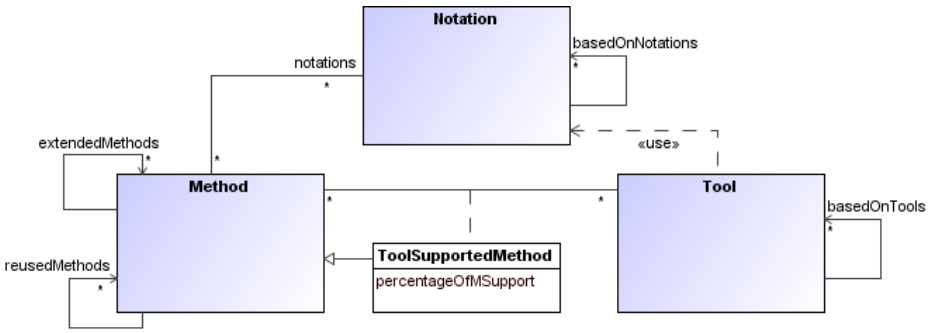

Fig. 8. Security Context: Connections between Tools, Notations and Methods

(cf. Fig. 5). Therefore, the class `KnowledgeObject` has associated names, tags and related sources, which could be any kinds of sources, as publications or URLs. We represent security issues, such as confidentiality, integrity and privacy by the class Security Property. The attribute `SecurityGoal`, which is denoted by a string, describes the goal of the property. For instance "integrity refers to the trustworthiness of data or resources" [37].

A Vulnerability is "a weakness that makes it possible for a threat to occur" [37]. Thus, it endangers security properties. Examples are XSS, SQL Injection, Buffer Overflows, etc. The objective of certain methods is to detect vulnerabilities or shield them from being exploited by a threat. Every vulnerability is located at least in one location (which is modeled as a UML enumeration). Furthermore, we include the categorization scheme from OWASP TOP 10 [9] (which is adapted from the OWASP Risk Rating Methodology [5]) using prevalence, impact level, detectability and exploitability. Regarding the latter two roles, the `Difficulty` "theoretical" means that it is practically impossible to detect or exploit a vulnerability (cf. Fig. 5).

A Threat is "a potential occurrence that can have an undesirable effect on the system assets or resources" [37, p.498]. We treat a threat as a kind of method which is vicious. At least one vulnerability has to be involved, otherwise a threat is not malicious (and the other way around), which is denoted by the multiplicity [1..*]. Additionally, threats can be mitigated by other methods.

4.4 Data Collection

High-quality data is the basis needed to obtain good evaluation results. Therefore we create a rigorous schema which describes a set of properties that have to be defined before starting collecting data. The model we build contains all the relevant features needed during data collection. It is a approach based on Kitchenham's systematic literature review [10]. Conversely to Kitchenham's approach, we do not restrict ourselves to reviewing literature; we also include in-

formation about tools which cannot always be found in papers, but on websites and on information which is obtained from benchmarks or experiments.

Data collection comprises among others a search process that can be performed in different ways, e.g., the search can be automated or not, or it can be a depth-first or a breadth-first search (c.f. Fig. 9). Depth-first means, that the aim of a search is to extract a lot of detail information about a relatively small topic, whereas a breadth-first search is good to get an overview of a broader topic.

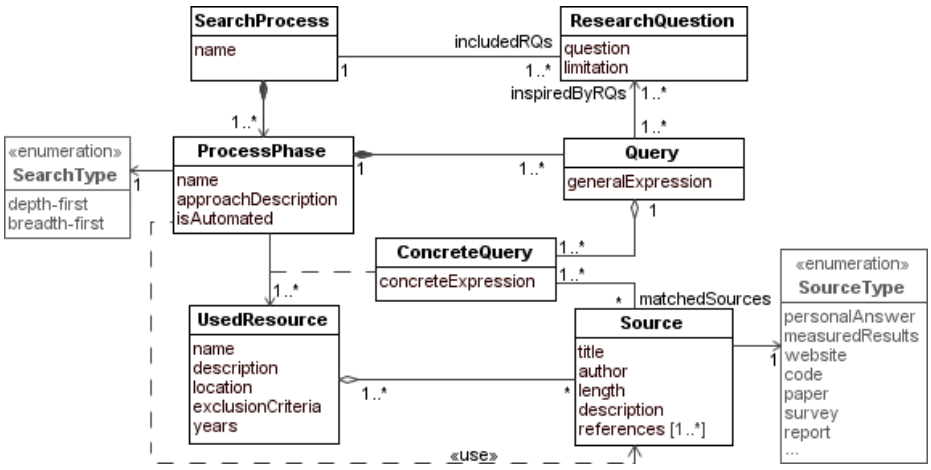

Fig. 9. SecEval: Data Collection [3]

Similar to Kitchenham's literature review, research questions are used to define the corner stones and the goals of the search. Please note that for us the term "research" does not necessarily refer to *scientific* research. Queries can be derived from research questions. As different search engines support different types of queries, concrete queries are specific for each resource, as e.g., Google Scholar. Queries can also refer to questions which are used as a basis for experiments (cf. Sect. 6). In addition, resources that will serve as data sources for the evaluation need to be chosen. If a concrete query matches sources, as papers, websites or personal answers, we classify the source at least by author and description (as an abstract) and provide information about the type of source and at least one reference where to find it. The process of data collection and data analysis is depicted in Fig. 3.

In Fig. 9 the use of an association class for `ConcreteQuery` (depicted by a dashed line) denotes that for each pair of `ProcessPhase` and `UsedResource`, the class `ConcreteQuery` is instantiated. The concrete search expression is derived from a general search expression.

For example, the general search expression could be "recent approaches in Security Engineering" and we want to ask Google Scholar and a popular researcher.

For Google Scholar we could use “"Security Engineering" 2012..2013” as a concrete search expression and the concrete expression for asking a researcher could read: “I’m interested in Security Engineering. Which recent approaches in Security Engineering do you know?”

4.5 Data Analysis

Data is collected with the purpose to obtain an answer to research questions based on the analysis of the data.

Figure 10 depicts relevant concepts for analyzing data. First, we have to specify which type of strategy we want to use. Are we limited to quantitative analysis or do we focus on qualitative analysis? Accordingly, one can later refer to Kitchenham’s checklists for quantitative and qualitative studies [10] to ensure the quality of the own answers to the research questions.

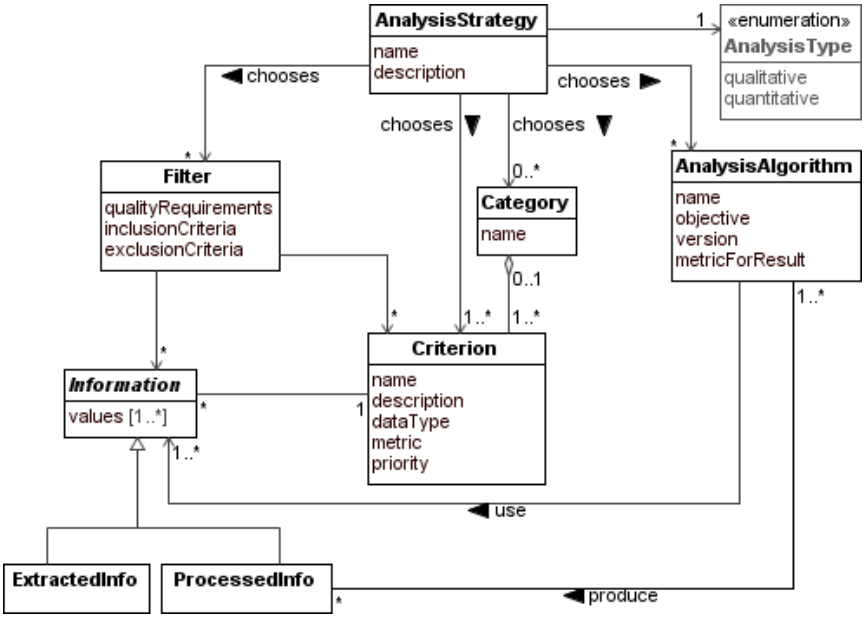

Fig. 10. SecEval: Data Analysis [3]

The analysis strategy defines which algorithm is employed and makes sure that the result of the algorithm fits to a criterion regarding meaning and metric. The algorithm does not have to be executable on a computer, but it might be implemented by a tool.

Criteria can be grouped by categories. A criterion gives more information about data values as it defines the data type (string, list of booleans, ..) and the

metric (milliseconds, ..). In addition, a priority can be defined which is useful when methods, tools or notations should be compared.

Information can be extracted from the sources which were found in the data collection phase (see ≪use≫ dependency starting from the class `ExtractedInfo` in Fig. 4), or they can be processed using an analysis algorithm.

For example, a relation `IsCompatible_NxN_ToolIO` can be seen as instances of an analysis algorithm. It expresses that "two notations are compatible if there exists a tool chain that can transform the first given notation into the second one" [35]. In this case, the algorithm might contain the depth-first search for a tool-chain consisting of tools where the output of one tool serves as input for the second one. The automation of such an algorithm is challenging, because in- and output of tools may differ.

Besides, a filter can be specified to disqualify results according to certain criteria as costs or quality. This filter is finer grained than the filter that is defined by `UsedResource`'s attribute `exclusionCriteria` used in the data collection, which only can be based on obvious criteria, such as the language the source is written in. In addition to this, the filter for data analysis accesses information as well as criteria and thus can exclude, e.g., methods, tools or notations from the evaluation that do not meet a high-priority requirement.

A valid question is how information, criteria and the security context model fit together. This is shown in Fig. 4: information can be stored in an instance of our security context model, which provides a sound basis when collecting data about methods, tools and notations. Consequently, the attributes `name` and `dataType` of a `Criterion` can be left blank when information is stored in an instance of our model, as attributes have a name and are typed. However, these attributes are needed when describing information which is not directly related to an instance of an artifact or not meaningful without their connection to a concrete analysis process.

Contrary to the context model, neither the collection of data nor the data analysis are security specific and thus can be applied in the same way to other domains.

5 Extensions of SecEval

As stated in the introduction, the core of SecEval cannot include all attributes which could be needed in the future. Therefore, SecEval's models are extensible, which means that users can add classes and attributes for their domain of research. In this section, we introduce an extension to show how SecEval can be enhanced in order to support OWASP's Risk Rating Methodology [5]. In addition, we provide an extension for Moody's method evaluation approach [6].

OWASP's Risk Rating. To rate risks for concrete IT systems is a common task for security engineers. OWASP's Risk Rating Methodology provides categories and terms for this task. Figure 11 depicts the extended model whereby added connections use thick line linkings.

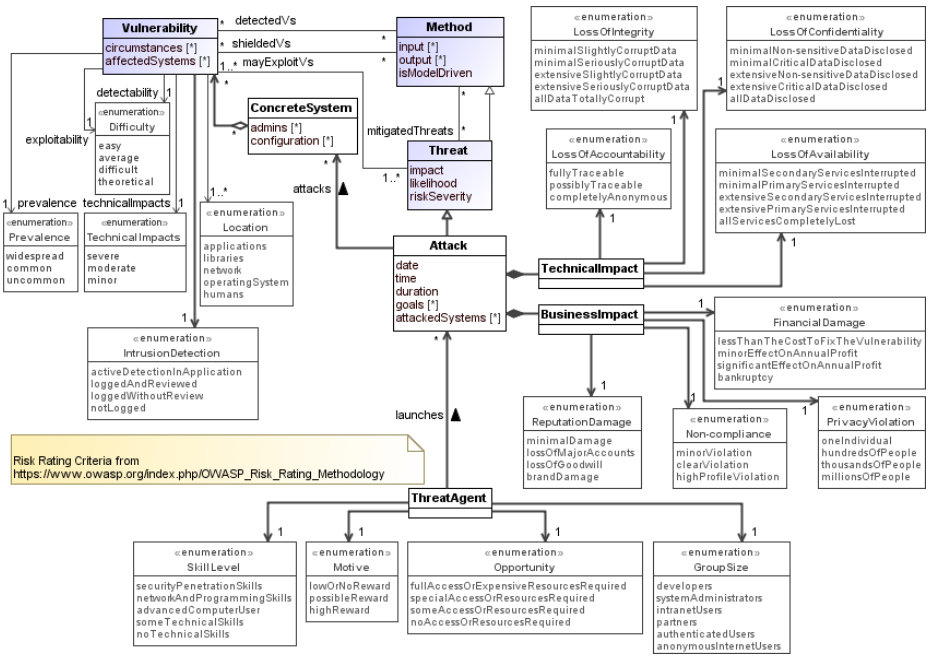

Fig. 11. Inclusion of basic risk evaluation approach

The class `Threat`, known from the basic context model, inherits its features to a concrete `Attack`. The severity of the risk (which is an attribute of `Threat`) can be calculated by likelihood multiplied with impact. The likelihood is derived from the factors which describe the vulnerabilities and the threat agents, whereas the impact is determined by the concrete technical and business-related consequences. Therefore, each enumerations' literal is mapped to a likelihood rating from 0 to 9. For more information the interested reader is referred to [5].

Moody's Method Evaluation Approach. Experimental approaches are used to evaluate the success of using a method in practice. Our extension of SecEval to express Moody's concepts is shown in Fig. 12: we introduce a `Test` class that is connected to at least one method and vice versa. The test uses the method on at least one example and is executed by `TestingParticipants`. Each participant assesses the method using Moody's evaluation criteria:

- "Actual Efficiency: the effort required to apply a method.
- Actual Effectiveness: the degree to which a method achieves its objectives.
- Perceived Ease of Use: the degree to which a person believes that using a particular method would be free of effort.
- Perceived Usefulness: the degree to which a person believes that a particular method will be effective in achieving its intended objectives.

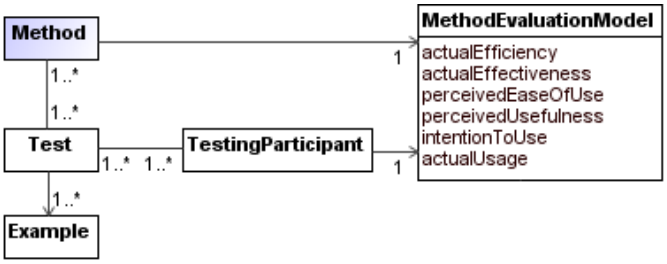

Fig. 12. Method extension using Moody's method evaluation approach

– Intention to Use: the extent to which a person intends to use a particular method.
– Actual Usage: the extent to which a method is used in practice" [6].

Usually, the average value of the participants' results is used as final evaluation result for the method under test.

6 Validation of the Evaluation Approach

The soundness of our SecEval evaluation approach is proved by a case study on security testing of web applications.

Web applications are the focus of many attacks. Thus, many methods such as "penetration testing" or "vulnerability scanning" are used to identify security flaws. These methods are supported by many commercial and open-source tools and frequently it is not easy to decide which one is the more suitable for the tests to be performed. In this section, we use our SecEval approach to evaluate vulnerability scanners for web applications.

Data Collection. According to the SecEval approach the first step consists of specifying the data that should be collected. This is done by an instance model as shown in Fig. 13, which depicts instances of the classes we have already defined in Fig. 9. For example, instances of the class `ResearchQuestion` define two research questions, a high-level and a concrete one. We used identical background colors for instances of the same classes and omitted all `name` attributes in case a name (e.g., `p3`) is given in the header of an instance.

Research question `q1` ("Which security-related tools and methods are available and how do they compare?", cf. Fig. 13) is very general. In the first process phase `p1`, 13 methods and 18 tools were selected [38]. More detailed information was gathered in the second process phase `p2` about: vulnerability scanning, penetration testing, fuzzing and the classification into black- grey- and white-box testing. Examples for tools are WSFuzzer, X-Create and WS-Taxi, just to mention a few. As we already added most of the methods and tools we found to the CBK [1], we focus on `q2` in this section.

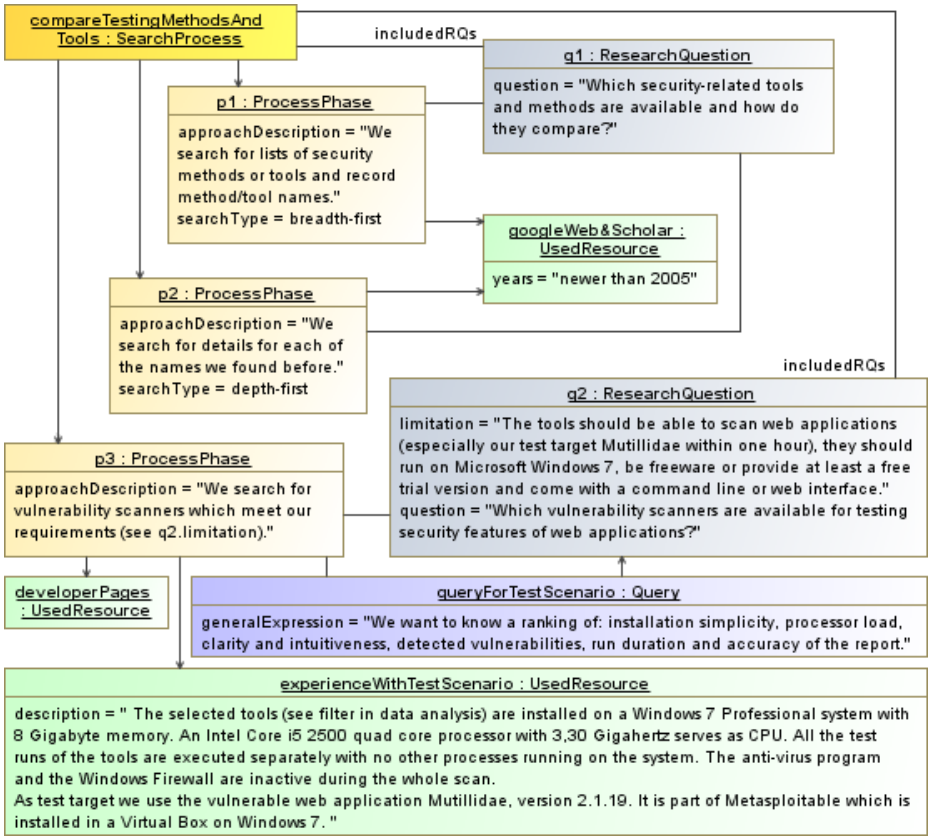

Fig. 13. Case Study: Data Collection

Research question q2 ("Which vulnerability scanners are available for testing security features of web applications?") is a typical question which could be asked by security engineers working in a company. The "sources" (i.e., tools) we selected for analysis were [39]: a) Acunetix Web Vulnerability Scanner[1], b) Mavituna Security - Netsparker[2], c) Burp Scanner[3], d) Wapiti[4], e) Arachni[5], f) Nessus[6], g) Nexpose[7] and h) Nikto[8].

[1] Acunetix. http://www.acunetix.com
[2] Netsparker. https://www.mavitunasecurity.com/netsparker
[3] Burp Scanner. http://portswigger.net/burp/scanner.html
[4] Wapiti. http://www.ict-romulus.eu/web/wapiti
[5] Arachni. http://www.arachni-scanner.com
[6] Nessus. http://www.tenable.com/de/products/nessus
[7] Nexpose. https://www.rapid7.com/products/nexpose
[8] Nikto. http://www.cirt.net/Nikto2

The instance `experienceWithTestScenario` describes how the data is gathered by testing the vulnerability scanners. Please note that SecEval does not impose the completion of the data collection phase before the data is analyzed. This means that the tests were partly executed on tools which were later classified as inappropriate. This becomes clear when we think of how evaluation works in practice: sometimes we have to collect a bunch of data before we observe information which, e.g., leads to the exclusion of a tool from the result set.

Data Analysis. The analysis phase consists in defining the analysis strategy and selecting a filter that enforces the requirements (`limitations`) defined for question `q2`. Figure 14 depicts instances of the data analysis model we defined in Fig. 10.

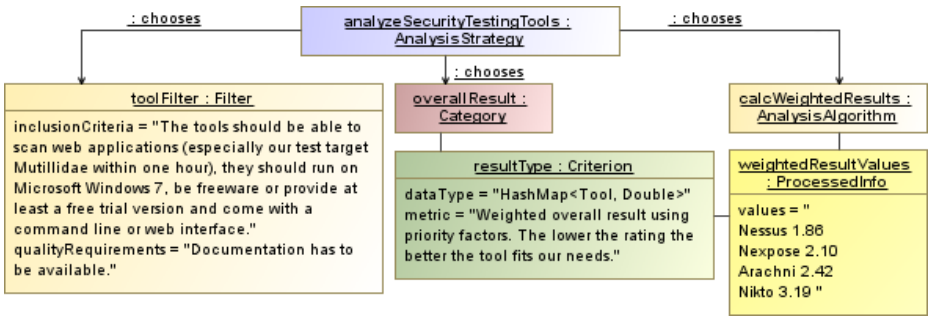

Fig. 14. Case Study: Data Analysis – Results [3]

Before going into detail about particular results of our experiments, we first take a look at the overall result regarding our research question `q2`. Figure 14 thus depicts an instance of the class `ProcessedInfo`, which is called `weightedResultValues`.

Only four tools passed our filter: Arachni and Nikto, which provide command-line interfaces and Nessus and Nexpose, which also provide web interfaces. From our list of tools from above, the trial of *a*) only allows to scan predefined sites. Tools *b*) and *c*) do not support a command line or web interface in the versions that are free. A run of tool *d*) on our test target Multidae[9] took six hours.

Apart from information available online, we experimented with the tools that passed the filter, in order to obtain data for our tool evaluation (`q2`). We evaluated the following criteria (and weighted them as indicated in the brackets, cf. `queryForTestScenario`):

– Installation simplicity (0.5)
  Do any problems occur during installation?

[9] NOWASP (Mutillidae). `http://sourceforge.net/projects/mutillidae`

- Costs (1)
  How much do the tool cost? Is it a one-time payment or an annual license?
- Processor load (1)
  How high is the CPU load while running the scanner
- Clarity and intuitiveness (1)
  Is the tool easy to understand, clearly structured and user-friendly
- Run duration (1)
  How long does a scan take?
- Quality of the report (2)
  How detailed is the report of the scan? Which information does it contain?
- Number of detected vulnerabilities (4)
  How many vulnerabilities does the tool detect on our test environment?

As we can see in Fig. 14, an algorithm is involved, which calculates results according to a rating. The rating is depicted in Fig. 15.

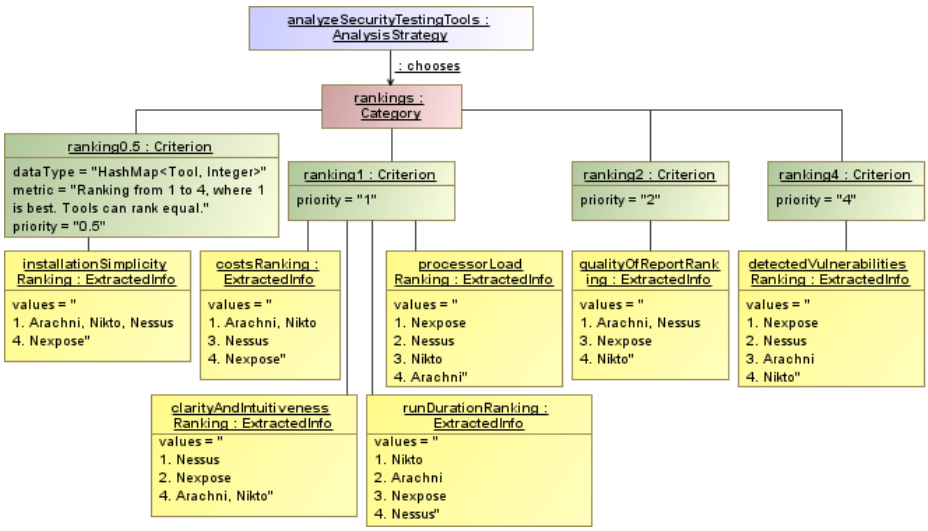

Fig. 15. Case Study: Data Analysis – Ratings

Lower factors of a criterions' priority denote that we consider the criterion less important. Table 1 contains the measured results as well as the average[10] and weighted[11] results.

In addition, we show how intermediate values of our tests could be described in our data analysis model in Fig. 16, as e.g., the costs of the tools or the operating systems it runs on. Concrete instances of `Information` classes are not depicted; the interested reader is referred to [39].

[10] AVG: average
[11] WAVG: weighted average according to ratings

Tool	Inst. Costs	CPU	Clarity	Time	Vuln.	Report	AVG[10]	WAVG[11]
Nessus	1	2	2	1	4	1	2	1,86	1,86
Arachni	1	1	4	4	2	1	3	2,29	2,42
Nexpose	4	4	1	2	3	3	1	2,57	2,10
Nikto	1	1	3	4	1	4	4	2,57	3,19

Table 1. Case Study: Final Tool Ranking (adapted from [39])

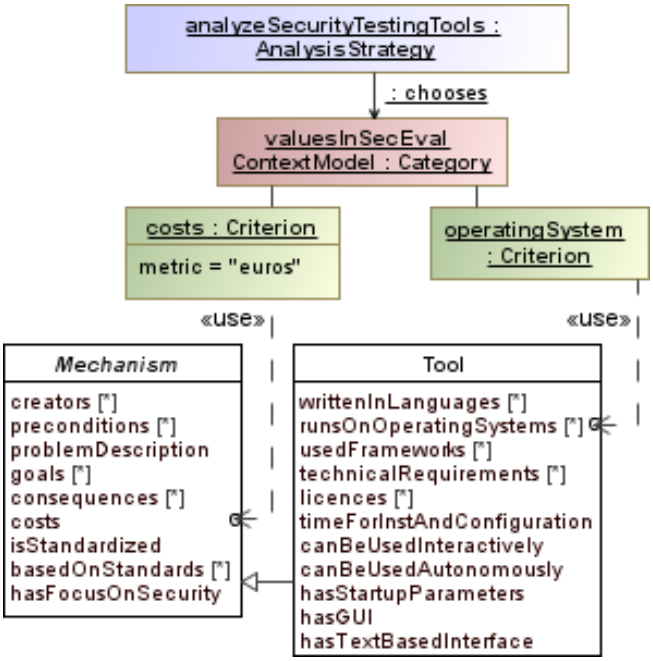

Fig. 16. Case Study: Data Analysis – Values

Security Context Model. We integrated all four tools into the NESSoS tool workbench, called SDE [40]. If they are executed from within the SDE, URL and port of the web application under test have to be provided by the user. To try out the vulnerability scanners it is possible to use Multidae as a target, as we did above. Multidae comes with Metasploitable[12], an intentionally vulnerable Linux virtual machine. Therefore, the default configuration for the integrated SDE tools point to a local Multidae instance, but can be changed at any time.

As SecEval's context model is more detailed, we modeled the context of vulnerability scanning of web applications and two of the tested tools: Nessus and Nikto. Figure 17 shows an instance diagram of the context model, which we have already depicted in Fig. 5. The association between vulnerabilities, as well

[12] Metasploitable. `http://www.offensive-security.com/metasploit-unleashed/Metasploitable`

as further supported methods are not depicted in Fig. 5; the interested reader is referred to the model example that can be downloaded [36].

The vulnerabilities that are modeled are the top 3 from OWASP's top 10 project 2013 [9]. Vulnerabilities may be caused by other vulnerabilities, for example invalidated input can lead to injection vulnerabilities.

We recommend using additional classes for extensions, such as a class to detail a test run, using attributes as run duration or processor load. Although building the instance model was straight forward, our experience with SecEval and the UML CASE tool MagicDraw showed us that the layout is not inviting to read the containing information. Consequently, we are looking forward to a future implementation of SecEval as a kind of semantic Wiki, as described in the following section.

7 Towards an Implementation of SecEval

Currently, we are implementing the concept of SecEval as a flexible web application. The aim is to provide a Wiki-like knowledge base for software and security engineers as well as for developers. Advantages of a web-based implementation of SecEval would be that connections to existing elements (like other methods or vulnerabilities), can be added without building the knowledge base from scratch and that data sets of previous evaluations remain available for future research.

The SecEval Wiki will support the following use cases: (1) viewing Knowledge Objects (knowledge objects), (2) editing knowledge objects, (3) importing external information and (4) searching for information to answer research questions, which can result in executing a tool-supported SecEval evaluation process.

Viewing knowledge objects. For the implementation of SecEval's context model, we experiment with a system that provides three views on each knowledge object:

- A *tabular view* that shows attributes' values, grouped by classes (presented as boxes) of SecEval's models. Which attributes are shown can be defined by the user. This view is especially useful for comparing knowledge objects.
- A *UML view* that presents an instance model of SecEval's UML model. The advantage of this view is that it is easy to examine links between several knowledge objects.
- A view that shows *continuous text*, enriched by boxes that can be placed between paragraphs or beside the text, similar to Wikipedia[13].

The user should be able to switch between these views at any time.

Editing knowledge objects. When creating a new element in the Wiki, the page is empty at first and shown in the *continuous text view* so that text can by written and structured by headings and paragraphs immediately, like in most

[13] Wikipedia. `https://www.wikipedia.org/`

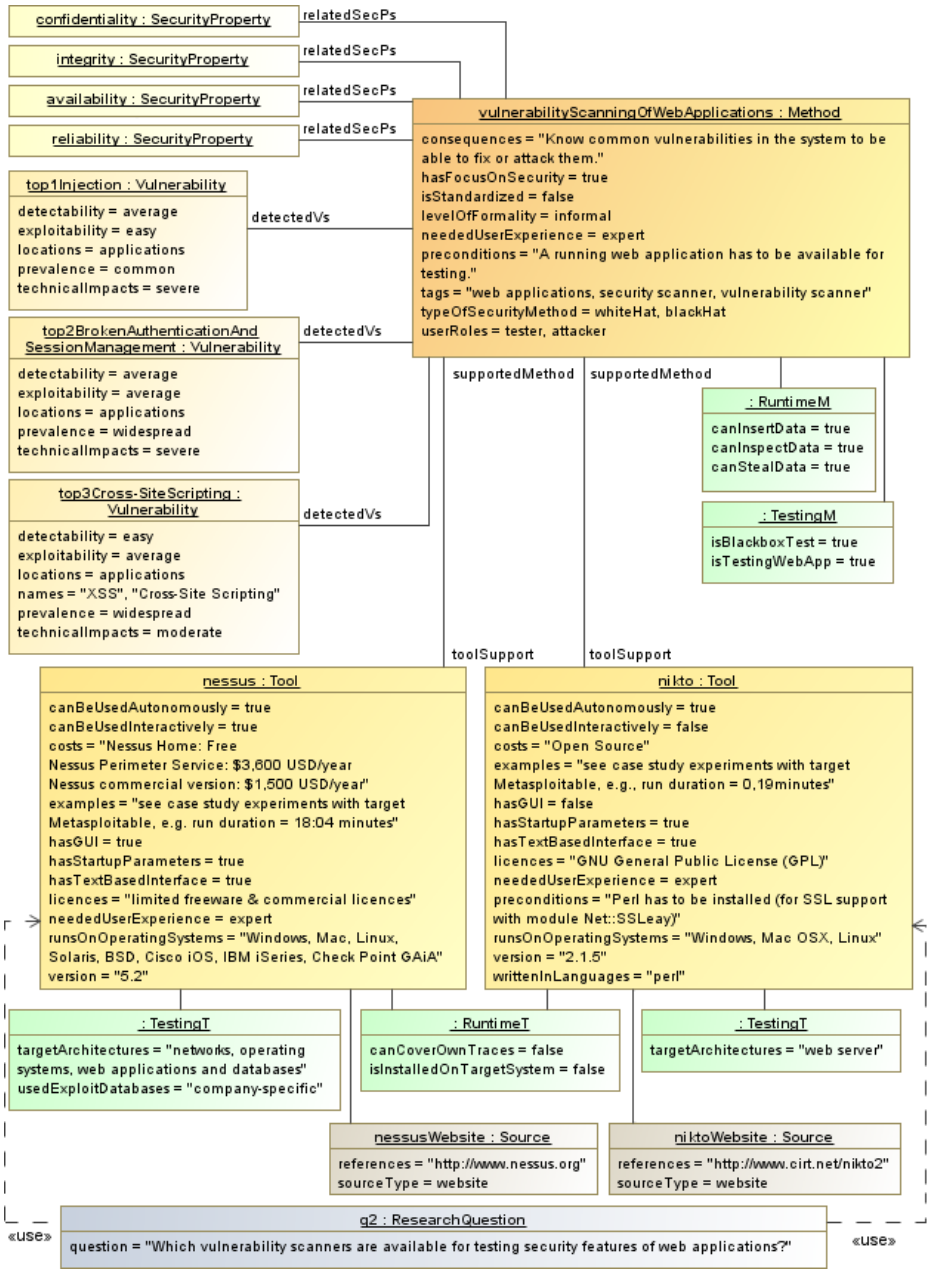

Fig. 17. Case Study: Instances of the Context Model (excerpt) [3]

Wikis. Additionally, on the side of the screen, common attributes are presented in a sidebar which can be dragged onto the Wiki page in order to fill them with actual values and to arrange them within the text or in boxes. These attributes correspond to attributes from SecEval models. For example, the user can specify some attributes of the `Tool` class, as technical requirements, licenses or the language the tool is written in.

Usually, information about knowledge objects has already been stored in continuous text form. In such a case, the application should allow to easily mark text and to click on an attribute on the sidebar. The attribute is then linked to the text so that it changes automatically when the text is altered. It is important to store previous versions not only for continuous text, but also for attributes, as both can easily be changed or deleted.

For the sidebar (which is resizable up to full-screen) it is also useful to implement different views, for example:

- a *UML view*, showing the full SecEval class diagrams for experts. This is the counterpart to the instance view for a concrete entry of the Wiki.
- an *auto-suggestion view* in which single attributes are shown according to an attribute-based suggestion system. This system can then recommend attributes which seem to be useful in the current context, as e.g., attributes of testing tools, as soon as it becomes clear that a user describes a tool from the domain of testing.

Recommendation includes that the system needs to explain rules inferred by SecEval, as e.g. that it is useful to describe a tool and a corresponding notation in two separate entries, even if the notation has only been used by this tool so far. A focus is on the connection between several knowledge objects in our SecEval system and on the possibility to add data which is not only associated with one knowledge object, as e.g., evaluation results.

Importing external information. Another useful feature is syndication, i.e. to be able to insert text from other web pages, as from Wikipedia or from vulnerability management systems, which are correctly cited and updated automatically. This task could be eased by step-by-step wizards and good attribute recommendation according to the attributes selected so far and the information provided. For example, if the user inserts the URL of a Wikipedia article, the article is displayed in a window that allows selecting passages and to transfer them to the SecEval system immediately, along with a linked cite.

Another requirement is the import of text from PDF files. Hereby, a challenge is to deal with licensed books or papers, because citing small passages is usually allowed, whereas publishing the whole document in the web is prohibited.

Searching for information to answer research questions. In addition to the implementation of SecEval's context model, the application should support the process of collecting and analyzing data to answer a concrete research question.

Simple questions can be answered using a full-text search. More complex questions can involve several knowledge objects and their attributes, so that the

search function has to be able to rely on the associations between knowledge objects stored in the knowledge base.

If the requested information cannot be found in the knowledge base, a wizard might suggest using SecEval's process to collect and analyze information. Ideally, the wizard allows jumping between several process steps while offering to record information for SecEval's data collection and data analysis models. The users can decide whether their research question should be public[14]. At the end of a complex evaluation process, artifacts as research questions, used sources and the concrete approach of a research can be published to save time and money in case a similar question will arise again in the future.

A general requirement for our implementation is the usability of the interface. For example, the CBK provides a complex search function, but it turned out that it is rarely used, because attributes have to be selected by using their technical, short names. For SecEval, it might be helpful to present descriptions and to suggest attributes according to a catalogue that learns how users tend to name a concept. Ideally, this search does not require a complex interface, but supports the user with auto-completion or wizards when typing a query into a text box.

8 Conclusions

We presented a conceptual framework, called SecEval, for the structured evaluation of so-called knowledge objects – methods, tools, notations, security properties, vulnerabilities and threats – in the area of secure software. SecEval is based on the structured literature review by Kitchenham et al. [10] and inspired by the C-INCAMI framework of Becker et al. [12], to name a few. Our approach is designed to ease the process of doing research or obtaining pragmatic answers in the area of security whether the research question aims at scientific or engineering issues.

SecEval is represented as a UML model and follows the separation of concerns principles. These concerns are:

- *An evaluation process* that specifies the set of tasks and information pieces needed to evaluate methods, tools and notation in the security area.
- *A security context model* for describing features of the security-relevant knowledge objects.
- *A data collection model* that records the way how data is gathered. It mainly comprises the research question, collection process, used resources and the queries for finding sources that might be used to answer the question.
- *An analysis model* which defines the analysis strategy and the filters and algorithms it uses on the collected sources. Furthermore, the data structure for information is exactly specified, regardless of whether the data is to be stored in the security context model or not.

[14] Discussions can also help to answer a research question, therefore it is desirable to connect the Wiki with question/answer systems as, e.g., Stackoverflow `http://stackoverflow.com/`.

An advantage of our context model is that it can describe tools and methods according to their placement within the software development life cycle. In case SecEval does not provide all attributes for expressing elements of related domains, it can easily be extended, as demonstrated for Moody's method evaluation approach and OWASP's Risk Rating Methodology.

To validate our SecEval approach, we performed a case study about methods and tools from the area of security focusing on a research question about the selection of vulnerability scanners for web applications. For the case study all elements were presented as UML objects. To improve the practicability of our approach, we envisioned how an implementation of a SecEval knowledge base might look like and which requirements might be important.

Summarizing, SecEval provides a sound basis for evaluating research questions related to secure software engineering. This might ease the process of doing research in the area of security no matter whether a research question aims at scientific or engineering issues. In the future we plan to evaluate further research questions using SecEval, describing knowledge objects that are security-related and those that are related to other domains. Besides, it would be interesting to implement SecEval as a smart and flexible knowledge base and to execute empirical studies to measure the utility of our framework.

References

1. CBK: Common Body of Knowledge. http://nessos-project.eu/cbk (2013)
2. NESSoS: Network of Excellence on Engineering Secure Future Internet Software Services and Systems. http://nessos-project.eu/ (2014)
3. Busch, M., Koch, N., Wirsing, M.: SecEval: An Evaluation Framework for Engineering Secure Systems. MoK'14 (2014)
4. Busch, M., Koch, N.: NESSoS Deliverable D2.4 – Second release of Method and Tool Evaluation. (2013)
5. OWASP Foundation: OWASP Risk Rating Methodology (2013) https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.
6. Moody, D.L.: The method evaluation model: a theoretical model for validating information systems design methods. In Ciborra, C.U., Mercurio, R., de Marco, M., Martinez, M., Carignani, A., eds.: ECIS. (2003) 1327–1336
7. Lipner, S., Howard, M.: The Trustworthy Computing Security Development Lifecycle. Developer Network - Microsoft (2005) http://msdn.microsoft.com/en-us/library/ms995349.aspx#sdl2_topic2_5.
8. ISO/IEC: 27001: Information technology – Security techniques – Information security management systems – Requirements. Technical report, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) (2013)
9. OWASP Foundation: OWASP Top 10 – 2013 (2013) http://owasptop10.googlecode.com/files/OWASPTop10-2013.pdf.
10. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report (2007)
11. Beckers, K., Eicker, S., Heisel, M., (UDE), W.S.: NESSoS Deliverable D5.2 – Identification of Research Gaps in the Common Body of Knowledge. (2012)

12. Becker, P., Papa, F., Olsina, L.: Enhancing the Conceptual Framework Capability for a Measurement and Evaluation Strategy. 4th International Workshop on Quality in Web Engineering (6360) (2013) 1–12
13. RWTH Aachen University: i* notation `http://istar.rwth-aachen.de/`.
14. Elahi, G., Yu, E., Zannone, N.: A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. Requirements Engineering **15**(1) (2010) 41–62
15. Wang, J.A., Guo, M.: Security data mining in an ontology for vulnerability management. In: Bioinformatics, Systems Biology and Intelligent Computing, 2009. IJCBS '09. International Joint Conference on. (2009) 597–603
16. RWTH Aachen University: SWRL: A Semantic Web Rule Language Combining OWL and RuleML (2004) `http://www.w3.org/Submission/SWRL/`.
17. Moyano, F., Fernandez-Gago, C., Lopez, J.: A conceptual framework for trust models. In Fischer-Hübner, S., Katsikas, S., Quirchmayr, G., eds.: 9th International Conference on Trust, Privacy & Security in Digital Business (TrustBus 2012). Volume 7449 of Lectures Notes in Computer Science., Vienna, Springer Verlag, Springer Verlag (2012) 93–104
18. Fernandez, C., Lopez, J., Moyano, F.: NESSoS Deliverable D4.2 – Engineering Secure Future Internet Services: A Research Manifesto and Agenda from the NESSoS Community. (2012)
19. Bertolino, A., Busch, M., Daoudagh, S., Lonetti, F., Marchetti, E.: A Toolchain for Designing and Testing Access Control Policies. In Heisel, M., Joosen, W., Lopez, J., Martinelli, F., eds.: Advances in Engineering Secure Future Internet Services and Systems. Volume LNCS 8431., Springer (2014)
20. Giorgini, P., Mouratidis, H., Zannone, N.: Modelling Security and Trust with Secure Tropos. In: In Integrating Security and Software Engineering: Advances and Future Vision. (2006)
21. A., D., van Lamsweerde A., S., F.: Goal-directed Requirements Acquisition. **20(1-2)** (1993) 3–50
22. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3) (2004) 203–236
23. Gedik, B., Liu, L.: Protecting Location Privacy with Personalized k-anonymity: Architecture and Algorithms. **7(1)** (2008) 118
24. Jürjens, J.: Secure Systems Development with UML. Springer (2004)
25. Basin, D., Doser, J., Lodderstedt, T.: Model Driven security: From UML Models to Access Control Infrastructures. ACM Trans. Softw. Eng. Methodol. **15**(1) (2006) 39–91
26. Basin, D., Clavel, M., Egea, M., Garcia de Dios, M., Dania, C.: A model-driven methodology for developing secure data-management applications. Software Engineering, IEEE Transactions on **PP**(99) (2014) 1–1
27. García de Dios, M.A., Dania, C., Basin, D., Clavel, M.: Model-driven Development of a Secure eHealth Application. In Heisel, M., Joosen, W., Lopez, J., Martinelli, F., eds.: Advances in Engineering Secure Future Internet Services and Systems. Volume LNCS 8431., Springer (2014)
28. Busch, M., Knapp, A., Koch, N.: Modeling Secure Navigation in Web Information Systems. In Grabis, J., Kirikova, M., eds.: 10th International Conference on Business Perspectives in Informatics Research. LNBIP, Springer Verlag (2011) 239–253
29. Busch, M., Koch, N., Wirsing, M.: Modeling Security Features of Web Applications. In Heisel, M., Joosen, W., Lopez, J., Martinelli, F., eds.: Advances in Engineering Secure Future Internet Services and Systems. Volume LNCS 8431., Springer (2014)

30. Goldstein, A., Frank, U.: Augmented Enterprise Models as a Foundation for Generating Security-related Software: Requirements and Prospects. In: Model-Driven Security Workshop in conjunction with MoDELS 2012 (MDsec 2012), ACM Digital Library (2012)
31. Busch, M., Koch, N., Masi, M., Pugliese, R., Tiezzi, F.: Towards Model-Driven Development of Access Control Policies for Web Applications. In: Model-Driven Security Workshop in conjunction with MoDELS 2012 (MDsec 2012), ACM Digital Library (2012)
32. Microsoft: Dafny. `https://research.microsoft.com/en-us/projects/dafny/` (2014)
33. Jacobs, B., Smans, J., Piessens, F.: VeriFast. `http://www.cs.kuleuven.be/~bartj/verifast/` (2013)
34. CORAS method: CORAS tool. `http://coras.sourceforge.net/` (2013)
35. Busch, M., Koch, N.: NESSoS Deliverable D2.1 – First release of Method and Tool Evaluation. (2011)
36. Busch, M.: SecEval – Further Information. `http://www.pst.ifi.lmu.de/~busch/SecEval` (2014)
37. Bishop, M.: Computer Security: Art and Science. 1st edn. Addison-Wesley Professional (2002)
38. Schreiner, S.: Comparison of Security-related Tools and Methods for Testing Software (2013) Bachelor Thesis.
39. Lacek, C.: In-depth Comparison and Integration of Tools for Testing Security features of Web Applications (2013) Bachelor Thesis.
40. Busch, M., Koch, N.: NESSoS Deliverable D2.3 – Second Release of the SDE for Security-Related Tools. (2012)