



SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY
INFORMATICS
TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

**Using Secure Software Engineering Metrics to
support the automated calculation and
visualization of Team Security Maturity in
Agile Development Projects**

Timo Zandonella



SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY
INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

**Using Secure Software Engineering Metrics to
support the automated calculation and
visualization of Team Security Maturity in
Agile Development Projects**

**Einsatz von Metriken zur sicheren
Softwareentwicklung für die Unterstützung der
automatisierten Berechnung und
Visualisierung des Team Reifegrads in agilen
Entwicklungsprojekten**

Author: Timo Zandonella
Supervisor: M.Sc. Sascha Nägele
Advisor: Prof. Dr. Florian Matthes
Submission date: 15.12.2022

I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 15.12.2022

Timo Zandonella

Abstract

New secure software engineering challenges have emerged for large-scale agile development, such as an increase in malicious attacks and subsequent data breaches. To address them, more security measures need to be taken, many of which fall under the responsibility of a central security governance unit. Due to the increased workload, security governance often cannot keep up with the agile pace of the development teams, which can result in development teams being limited in how efficiently they can develop. One way to increase efficiency and security levels in development teams is to give security-capable teams more freedom and autonomy in their security decisions. Measuring a team's security capability can be achieved using security maturity scores, which systematically assess a team and its processes in different areas. Assessing often relies on manual self-assessments by team members or external audits by security specialists in the form of questionnaires, which can be skewed and time-consuming to complete.

We propose secure software engineering metrics as a complementary team security maturity tool to address the existing issues with assessments. Secure software engineering metrics measure team-wide or product-specific attributes, such as the number of unmitigated vulnerabilities or the team's knowledge of security policies. Measurement is continuous and occurs at all stages of the software development life cycle. However, there is no recognized collection of high-quality metrics, and those proposed to date are often described in an unstructured way. In addition, existing security maturity models do not use secure software engineering metrics to calculate a security maturity score.

To address this research gap, we first create a structured catalogue of security metrics. We establish rigorous qualification criteria for the metrics and collect the information using a standardized catalogue format for software metrics. Additionally, we research which tools can automatically measure the catalogued secure software engineering metrics. Second, we integrate the cataloged metrics into a maturity model to complement the (self-)assessments and propose a procedure to calculate a maturity score from the assessments and the secure software engineering metrics. Third, we implement a prototype web application that enables a development team and decision makers to track security metrics and their maturity throughout the software development lifecycle.

In this work, we first briefly explain previous work on security metrics. We then present the systematic literature review we conducted and its result, our proposed metrics catalogue. Afterwards we depict the integration of security metrics into a security maturity model for teams and describe the implemented prototype. To validate the functionality of the prototype, we conducted eight interviews with industry experts. Generally the prototype was evaluated as beneficial, a major evolution to the original team security maturity model, and a helpful guidance tool during agile development. Finally, we summarize our work and key findings and provide starting points for future work.

Contents

Abstract	iii
1. Introduction	1
1.1. Motivation	1
1.2. Research objectives	3
1.3. Research approach	4
2. Foundations	5
2.1. Software security in agile software development	5
2.1.1. Agile software development	5
2.1.2. Security during the SDLC	6
2.2. Team security maturity	7
2.3. Secure software engineering metrics	8
2.3.1. Qualification criteria for security metrics	9
2.3.2. Assessing metrics with quality scores	9
2.3.3. Tools for metric measurement collection	10
3. Methodology	12
3.1. Systematic literature review	12
3.2. Interviews	15
4. Related Work	17
4.1. Security metrics catalogue	17
4.2. Security metrics for team security maturity	18
5. Security metrics	20
5.1. Catalogue structure	20
5.2. Team security metrics	21
5.2.1. Knowledge domain	21
5.2.2. Effort domain	22
5.3. Product security metrics	23
5.3.1. Analysis Domain	23
5.3.2. Design Domain	23
5.3.3. Implementation Domain	24
5.3.4. Deployment Domain	25
5.3.5. Maintenance Domain	25
5.4. Measuring security metrics with security tools	26

6. Team security maturity through security metrics	28
6.1. Security metrics in TSMM	28
6.2. Calculation of the overall security maturity score	30
7. Tool-supported security maturity	33
7.1. Use cases	33
7.2. Backend application	34
7.2.1. Security metrics collection	34
7.2.2. Security maturity calculation	36
7.3. Frontend application	39
7.4. Evaluation	45
7.4.1. Security maturity dashboard	45
7.4.2. TSMM data sources	47
7.4.3. TSMM approach	47
8. Discussion	49
8.1. Key finding and artifacts	49
8.2. Limitations	51
8.2.1. Validation of research methods	51
8.2.2. Validation of proposed software prototype	52
9. Conclusion and Future Work	54
9.1. Summary	54
9.2. Future work	55
A. Interview Questionnaire	56
B. Complete Catalogue of Security Metrics	57
B.1. Team security metrics	57
B.1.1. Knowledge	57
B.1.2. Velocity	58
B.2. Product security metrics	61
B.2.1. Analysis	61
B.2.2. Design	62
B.2.3. Implementation	65
B.2.4. Deployment	68
B.2.5. Maintenance	72
B.3. SMMM Calculation	74
List of Figures	76
List of Tables	77
Bibliography	78

1. Introduction

In this introductory chapter, we discuss the basic motivation and relevance for creating a catalogue of secure software engineering metrics (security metrics), the extension of a maturity model to measure security capabilities in an agile development team, and the need for software-based computation and visualization of a team security maturity score. From this motivation, we derive our research goals, which are structured by three research questions.

1.1. Motivation

Rising security ownership tension in (scaled-) agile development

Within large software engineering companies many agile development teams exist across different dimensions, e.g. across various products, business units or countries. In these organizations, the integration of secure software engineering practices is becoming increasingly important as the number of malicious attacks and security breaches increases year over year, as seen in Figure 1.1.

As the risk of a successful attack increases, so does the risk of large civil fines, private litigation, loss of customer trust, and loss of enterprise value. To implement, execute and monitor software security processes across heterogeneous teams, organizations often have a central IT governance and security unit responsible for these tasks. Once this unit is established, the question of process responsibility quickly arises: which processes are handled decentrally by the teams and which belong to the central unit? One possibility is for a team to report to the central unit every time it wants to deploy an artifact to the public domain. This would lead to an unmanageably high workload in the central unit and long idle times in the development teams. In contrast, a development team could also have complete freedom to deploy artifacts, and the central unit would only periodically pull an artifact from the team and perform a security analysis. This would solve the problem of workload in the central unit and development teams, but on the other hand would lead to many untested and potentially insecure artifacts in the public domain. It becomes clear that the ideal solution must lie somewhere in the middle.

Team security maturity resolves tension in multiple ways

To resolve the described ownership issues we need to assess on the one hand what security competencies lie within the development team and on the other hand which security processes can't be handled by the team and need to be executed by a central governance and security unit. To assess this split of security responsibilities different research and literature describe a

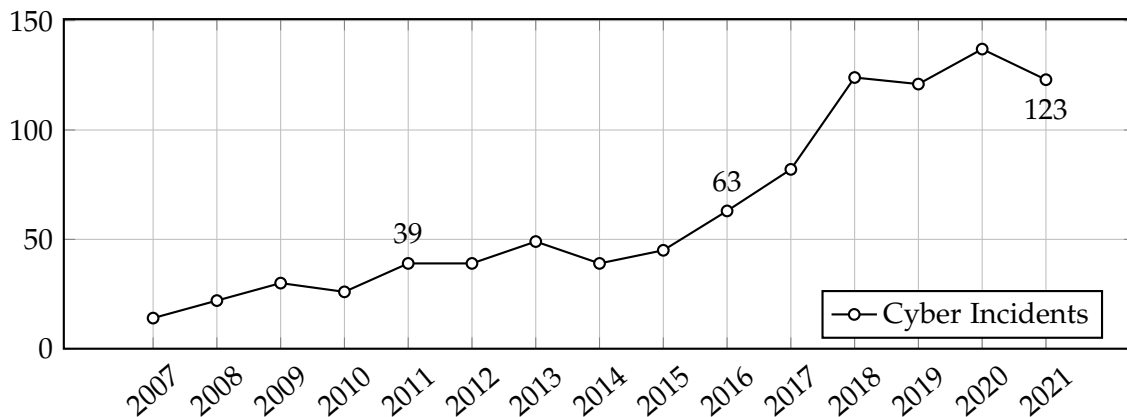


Figure 1.1.: The development of cyber incidents with losses of more than a million dollars [32]

possible solution: the introduction of team security maturity levels. The different security maturity levels describe the capability of an agile development team to develop secure and security-compliant software: if a team is more mature (e.g. if it has a higher maturity level), it is more capable. For the various stakeholders, assessing your team's security competency can have multiple benefits in the agile development process. Firstly, according to the assessed maturity level a team can be granted a specific level of autonomy by the central IT governance and security unit. By assigning a team a specific set of tasks to do themselves autonomously, the above described ownership conflict can be resolved, as it is clear which stakeholder is responsible for which tasks. This application is described in detail by Schenk [115]. Furthermore, assessing the maturity levels of teams that work on a specific product, a decision maker in the organization can better understand and predict how secure a new product release probably is. This improves the decision quality and can reduce the risk of successful security attacks. Finally, knowledge about the security level of the deployed applications in a potentially very large organization is essential. With maturity levels this organizational knowledge need can be fulfilled in a standardized way across any organizational structure. The team security maturity model (TSM) describes the composition of a maturity calculation, which is described in detail in Section 2, and additionally, it describes the process of capturing the maturity within the teams, which is through self-assessments.

Limitations of using (self-)assessments to deduce team security maturity

However, relying on self-assessments of team security maturity has its limitations and can fundamentally undermine the benefits described above. In practice, useful self-assessment questionnaires require many topics and are therefore time-consuming and labor-intensive. This can even lead to lack of acceptance by the development team and to hasty and inaccurate responses, as research has shown [144, p.68], [24, p.349],[87, p.M203]. Second, self-assessments

are inherently subjective and can lead to misperceptions. Two different team members might give different answers to the same self-assessment question. Finally, self-assessments by their nature can only capture the security status at regular intervals (i.e. once per sprint, once per month, once per quarter). A more precise measurement within the defined time period, e.g. immediately prior to the introduction of an artifact or shortly after the introduction of a new security tool, is not possible. With these limitations in mind, it becomes clear that calculating a maturity level through self-assessments can be inaccurate and thus undermines the fundamental goals of the maturity model described above.

1.2. Research objectives

In light of the general considerations regarding the limitations of self-assessments for assessing team security maturity, we introduce the possibility of measuring team security maturity with security metrics. Consequently, our research hypothesis is: *Using security metrics can lead to a more accurate assessment of team security maturity.* Three research questions arise from this research hypothesis:

Research question 1 (RQ1): Which security metrics exist and how can they automatically be captured with the support of security tools?

In order to introduce security metrics into the computation of a team's security maturity, we must first create a catalogue that can be used to collect structured information about each metric. This catalogue is the first artifact of the research. To create it and subsequently answer the first part of RQ1, a systematic literature review was conducted. To answer the second part, we created a matrix in which the identified security metrics are assigned to different security tools that enable their measurement.

Research question 2 (RQ2): How can security metrics be used to assess the security maturity of an agile development team?

To answer the posed question, we introduce the defined security metrics into the TSMM by adapting and extending its general structure and captured information. The first step is to add the security metrics to the model. Secondly, we extend TSMM to derive an overall maturity score from the maturity levels of each topic or security metrics and domain. Together, this yields the second artifact of the research, namely an adapted version of the TSMM extended by security metrics.

Research question 3 (RQ3): How can a team's security maturity be calculated, represented and visualized in a self-assessment tool?

With the third research question, we attempt to bring the theoretical artifacts into practice. This is done by extending an existing self-assessment tool with the new feature of measuring security metrics from a team's build pipelines, calculating the team's security maturity scores, and visualizing the results in a data dashboard. It becomes clear that the basis for this functionality lies in our responses to RQ1 and RQ2. Finally, we validate the value of the prototype with industry experts through the conduction of semi-structured interviews.

1.3. Research approach

To find an answer to our research questions defined in Section 1.2, we apply design science, in particular the design science research process (DSPR) of Peffers et al which is an appropriate model for several reasons [99]. First, it was developed specifically for research in information systems, our research area, and is widely used in this field [99, p.46, p.74]. Moreover, the use of DSPR helps us to be consistent with the earlier research of Watzelt [144]. It consists of six phases, namely:

1. Problem Identification
2. Objective
3. Design and Development
4. Demonstration
5. Evaluation
6. Communication

The communication of this research through this publication also follows this structure. First, the initial stages of problem identification and goal setting are described in the Introduction and Fundamentals chapters. Second, the work in the following two phases of the DSPR - namely, design and development and demonstration - is described in detail for each proposed answer to the research questions in three separate chapters 5, 6, and 7. Finally, the evaluation conducted is described in these chapters as well as in 8. Phase 6 of the DSPR, communication, is therefore completed by this publication.

2. Foundations

The following chapter describes the theoretical basis for the research. To set the research space, the general concepts of agile software development and secure software engineering are first identified. Then, in Section 2.2 the concept of maturity and specifically team security maturity is described. Finally, security metrics and their attributes are presented at a finer granularity.

2.1. Software security in agile software development

2.1.1. Agile software development

Agile software development is a management approach to software development, where

- small increments of software are released frequently
- any requests for change can be responded to quickly
- and the work happens in a self-organizing manner [40, p.1214], [12].

In 2021 over 94% of software engineering enterprises practice some form of agile software development [38, p.6]. This overwhelmingly large adoption rate founds partly on the different agile methods and frameworks that exists for all possible software engineering environments. Introducing agile methods was found to have large benefits to the velocity and quality of software, as visible in multiple studies (e.g. [69, p.379], [66, p.833]). For these reasons one could draw the conclusion that they are fundamentally necessary for software enterprises to stay in competition.

Our research focuses on a special agile environment scenario, which is a large-scale software development enterprise. We define large as having a team size of over 50 people in at least six teams, following the definition by Dikert [39, p. 88] and the usage of the definition by Watzelt [144]. Because of the scale of these large software engineering environments it is important to have standardized and reliable software production and delivery procedures during the software development life cycle (SDLC) [39, p.101], [44, p.3]. Our definition of a SDLC follows the version of the Unified Process Model by Jacobson [55] with five phases. In Figure 2.1 a visualization of the SDLC and its phases is displayed. This structure will be used throughout the research. Implementing the SDLC especially in a large enterprise can be done by following the development operations (DevOps) philosophy and approach, which describes how the SDLC is implemented in regard to different supporting tools or information exchange between people and teams [107].

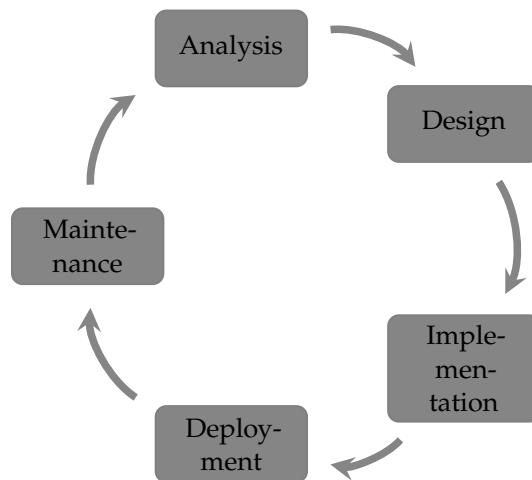


Figure 2.1.: The five phases of the SDLC

2.1.2. Security during the SDLC

The concept of software security describes the protection of information confidentiality, integrity and availability against intentional attacks by unauthorized parties [73, p.1]. When developing complex applications, it should not be an afterthought, but should be deeply integrated into the agile development process and SDLC to ensure sufficient security [80, p.1164], [108, p.2]. Early and accompanying security efforts can increase efficiency, reduce cycle-time and reduce costs [1, p.6], [29, p.531], [94, p.5]. An increase in the security activities implemented often leads to more secure software artifacts, which can increase customer trust in and satisfaction with the software [33, p.23]. Better security efforts may even result in the ability to enter new markets or gain a competitive advantage, i.e. in security-relevant applications in the healthcare or public sectors, which are only accepted if they meet very high regulatory security requirements. [127].

To reach a satisfying security level and practice security engineering, many activities during the SDLC need to be executed. These activities are describes in development security operations (DevSecOps), which as the name suggests is an extension of the traditional DevOps approach and extends it with security topics. DevSecOps activities are deeply integrated in the entire SDLC: At the beginning the security requirements are analyzed, the risk and risk tolerance of the developed application is determined and afterwards in the design and implementation phases the developed software should always adhere to the set risk tolerance [107]. Later in the cycle in subsequent phases such as deployment and maintenance automated tools can be used to automatically detect potential security issues [107]. DevSecOps is an important part of the shift security left trend, which describes that security aspects should be handled as early as possible in the SDLC, to reduce the amount of complicated and costly security efforts that need to be taken at a later stage [74].

Different teams, business units and companies implement DevSecOps vastly differently, by choosing different communication strategies, committing for different security processes or

different security tools. Measuring and evaluating the various actions taken is critical, as otherwise it is not possible to understand the success of the efforts made and improve the security posture [26, p.265]. One possibility to evaluate the success of the taken approaches is by calculating the team security maturity.

2.2. Team security maturity

Team security maturity describes an approach to assess the capability of an agile development team to develop secure and security-compliant software [144, p.15]. Our research focuses on the TSMM, which structures the development team information in a grid within different security domains that encapsulate relevant security topics. Assessing the capability has multiple benefits. The achieved capability can reduce the need for security governance in a central governance unit and empower a team's autonomy [144, p.80]. In addition, it can guide the development teams efforts, as the maturity scores include information about underperforming and satisfying security domains (see Section 6.2). The existing literature identified a multitude of usages for maturity levels for the different stakeholders in an agile development project and enterprise. Starting at the lowest aggregation in the development team itself, a maturity ranking can be used to predict how secure a software iteration is and with what confidence we can make this assumption [138, p.6], [106, sec. 5]. On a business unit level the maturity level can additionally be used to award teams with high maturity a higher degree of autonomy as extensively described by Schenk [115], which can reduce the need for costly and inefficient outsourcing of security processes while still maintaining a high level of security [115], [129, p.2]. And on an organizational level maturity level can additionally be used to get a broad organizational security overview, which can be used for adoption of the corporate security efforts or in an ideal case to exploit new security-conscious markets as described in Section 2.1.

Information about the team's security level is collected with three pillars: self assessments, external assessments, and automated. The first two pillars are assessed in the form of questionnaires, by team members or respectively by external auditors. The automated pillars includes suggestions on how to automatically measure issues with security tools. However, Watzelt intentionally did not include structured information to the extent of the other two pillars to allow for more flexible selection of security tools by the organizations using the TSMM [144, p.55].

Each topic that is described in the assessment pillars includes a short description, more information or an example, and the four maturity levels "No", "Partly", "Largely" and "Fully" on which it needs to be assessed. In Table 2.1 an exemplary topic from the self assessment pillar is displayed.

While assessments certainly allow to collect data about all the team security efforts and the products security level, there are also concerns about using them [65, p.272], [128, p.5]. Firstly, assessments can be time-consuming as they need to be quite extensive to give a good picture [100, p.83], [86, p.4]. In total the TSMM includes 38 topics in seven security domains, that need to be assessed, and each topic itself includes four different maturity

Name		Information	
We peer review our security documentation.		For example, we look at the documentation during code review to ensure it is sufficient.	
No	Partly	Largely	Fully
We have never reviewed any documentation.	For important security documents, we sometimes perform reviews, but only briefly.	We review the documentation together (at least 2) but often remain superficial.	We have ensured that at least two people sufficiently review the quality documentation for all security-related components.

Table 2.1.: An exemplary TSMC topic [144, p.104]

level descriptions. Secondly, they allow for subconscious or malicious misjudgements by participants [87, p.203]. Finally, because of their nature they only give periodic rather than continuous insights and therefore adoptions by the team between assessments cannot be measured. To mitigate these issues we integrate security metrics into team security maturity.

2.3. Secure software engineering metrics

Security metrics are quantifiable measurements used for assessing security related product imperfections and the teams' security efforts during the SDLC [58], [26], [57]. With security metrics uncertain and subjective aspects of software engineering are made definite and quantifiable [10, p.8]. They can eliminate doubt about the status-quo and the quality of security decision-making can be improved, which qualifies them as a valuable tool for security information collection [10, p.9], [62, p.9], [60, p.82]. Some aspects of security metrics also indicate that they can be often more suitable than assessments. Firstly, they can be continuously measured by security tools without much manual effort, which is described in detail in Section 2.3.3. Secondly, they do not have any subjective component and are therefore often a more consistent and comparable [104, p.3]. Finally, they can demonstrate the extent to which regulatory security requirements, such as certain ISO standards, and product security requirements, such as securing software endpoints, are met [50, p.626]. It is therefore often beneficial to teams and enterprises to introduce and measure security metrics continuously. In itself the concept of security metrics is not new, in fact research and application was conducted as early as the 1980s [140, p.88]. However, as security becomes more important as shown in Figure 1.1, the approach has become more popular again as a supporting tool for DevSecOps.

2.3.1. Qualification criteria for security metrics

An universally applicable selection of security metrics doesn't exist, as there is no recognized industry standard for describing security metrics, there is often no understanding of what metrics can be used for and security metrics are heavily reliant on context [85, p.102], [3, p.2]. As a result, many new and specialized metrics have been developed that are often of low quality and not transferable to other contexts [85]. For these reasons, Jaquith has established five rules that define what characteristics a useful software metric must meet [58].

According to these rules, a metric needs to be

- consistently measured without subjectivity
- cheap to gather
- expressed as a cardinal number or percentage instead of ordinal labels on a scale
- expressed using at least one unit of measure
- contextually specific showing relevancy to decision-makers, so they can act accordingly

As presented security metrics have different levels of quality depending on different metric attributes, but only high-quality metrics are suitable for decision-making [43, p.8], [58], [3, p.2]. Therefore, in order to create a relevant and applicable catalogue of security metrics, a qualification procedure must be developed and applied to candidates. The procedure used in the research applies Jaquith's presented criteria, and is described in detail in section 3.1.

2.3.2. Assessing metrics with quality scores

As shown in Section 2.3.1, security metrics are of different quality and relevance to decision-makers. Therefore, different frameworks have been developed to assess the quality and value of metrics, such as the security metrics maturity model (SMMM) by Muthukrishnan and Palaniappan [85]. It can be used to score metrics for a total of seven quantitative and qualitative fields. For each attribute, a metric can be assessed on an ordinal scale of 0, 0.5, or 1. The scores are then combined into a single percentage and a high percentage implies the metric is meaningful. For illustration purposes, the calculation for an exemplary metric is shown in Table 2.2.

Security Test Failure Rate		(87.5% + 83.33%) / 2 = 85.4%	
Quantitative		Qualitative	
Quantifiable	1.0	Measurability	1.0
Readiness	1.0	Meaningfulness	0.5
Repeatable	1.0	Correctness	1.0
Cardinal	1.0		
Score	3.5/4.0 (87.5%)	Score	2.5/3.0 (83.33%)

Table 2.2.: An exemplary calculation with the SMMM framework

2.3.3. Tools for metric measurement collection

Many security metrics can be measured semi- or fully automatically during the SDLC using security tools capable of measuring various static and dynamic aspects of the software artifact [130, p.78]. There are four main categories of software security tools, namely static application security testing (SAST), dynamic application security testing (DAST), interactive application security testing (IAST), and software composition analysis (SCA), which directly measure attributes of the software artifact [137, p.2]. In addition, security metrics can often be measured by security tools that indirectly assess security by collecting and aggregating metric measurements from different security tools [48, p.1]. We include two tool categories with this functionality: security information and event management (SIEM) and vulnerability management tool (VMT). In the following list, we explain the purpose, advantages, and disadvantages of each tool category with respect to measuring security metrics, and provide an example tool from each category.

- As the name suggest **SAST** performs static analysis of the written code without executing it [147]. It can therefore be used in early stages of the SDLC and is mainly able to uncover security vulnerabilities and possible attack vectors such as a race condition or a buffer overflow [23, p.91], [18, sec.5.2.1]. Generally tools are relatively easy to set up and interpretation of the returned security metrics measurements is straightforward, but many tools report false-positive vulnerabilities without actually requiring action [118, p.471].
→ Example Application: SonarQube¹
- **DAST** is an approach which involves running an application and attempting to attack it from an attacker’s perspective [98, p.558]. It is therefore able to find problems during runtime that cannot be detected with SAST tools, such as authentication and network configuration errors [118, p.472]. But because of this, analysis also cannot be performed at early stages of development, and interpretation of security metrics measurements can generally be more complex and time-consuming [118, p.472], [135, p.25859].
→ Example Application: OWASP ZAP²
- **IAST** combines the ideas of SAST and DAST to analyze whether a vulnerability is a false positive or can actually be exploited [133, p.1]. IAST tools can continuously monitor applications without regular scans and, unlike DAST tools, can be easily integrated into a DevSecOps pipeline [133, p.2]. However, it is a relatively new concept and therefore not as well researched in academia or as widely used in industry as other approaches [139].
→ Example Application: Seeker³

¹<https://www.sonarqube.org/>, accessed August 2022

²<https://www.zaproxy.org/>, accessed August 2022

³<https://www.synopsys.com/software-integrity/security-testing/interactive-application-security-testing.html>, accessed August 2022

- **SCA** is used to identify the potential security vulnerabilities and risks from open-source dependencies used in an application [126]. Open-source components can be insecure for example because they are no longer maintained and outdated, there exists an unintentional security-relevant bug in the open source dependency or it itself includes an insecure dependency [67, p.92], [126].
→ Example Application: Dependency Track⁴
- A **VMT** manages the applications security vulnerabilities with the help of different security tools. Often, VMT contain a list of unmitigated vulnerabilities, their severity, and their origin. The source for this information are vulnerability databases such as the “Common Vulnerabilities and Exposures” list⁵.
→ Example Application: Defect Dojo⁶
- **SIEM** systems are used to continuously monitor security events during production, i.e. through log files [92, p.519]. When a SIEM detects an event and security experts confirm that the event is malicious, a set disaster recovery plan that reduces the damage to the compromised assets should be carried out [15, p.35].
→ Example Application: OSSIM⁷

Additionally some security metrics can be measured with a project management tool (PMT), which is a tool that can track development tasks, issues, velocity and documentation during development [93, p.19]. An example for a PMT is the proprietary software Jira⁸ or the open source security requirement management tool OpenRat⁹. In particular, metrics that deal with the speed of security development or the quality and compliance with defined security requirements can often only be tracked with it. We therefor include PMTs in our research.

⁴<https://owasp.org/www-project-dependency-check/>, accessed August 2022

⁵<https://www.cve.org/>, accessed December 2022

⁶<https://www.defectdojo.org/>, accessed November 2022

⁷<https://cybersecurity.att.com/products/ossim>, accessed October 2022

⁸<https://www.atlassian.com/jira>, accessed August 2022

⁹<https://owasp.org/www-project-securityrat/>, accessed December 2022

3. Methodology

In our research we use two different research methods, which we describe in the following sections. Furthermore, we discuss the approaches and explain the theoretical background behind the research decisions taken.

3.1. Systematic literature review

To create a catalogue of security metrics a structured and systematic literature review was conducted. In short, the goal of the review was:

1. Identifying existing security metrics in the literature
2. Collecting structured information about the different identified security metrics

The review was conducted in a structured manner following the first two phases described by Webster and Watson [145]. We chose this approach because it has a focus on information systems and fits our research. Firstly, we choose a selection of literature search engines and define the used search strings. Secondly, we conduct a semi-automated search of literature with the defined parameters. Finally, we applied a qualification procedure to the found metrics to filter out lower quality metrics.

Phase 1: Foundation

In total the literature review was conducted within seven search engines, as no single source can include all relevant publications [21, p.120]. This selection of the used sources is derived from other literature reviews on security metrics [47, p.245], [21]. Three of the sources are direct literature publishing libraries, namely IEEE, ACM, and Science Direct (category ①). Scopus and Google Scholar were selected as literature search engines, as they collect publications from various sources and make them available at a central access point (category ②). Finally, the TUM Online Publication Access Catalog¹ and a normal Google search was performed to collect non-peer reviewed publications (category ③). For the identified publications we also define two qualification criteria, to make the research linguistically feasible and only include up-to-date literature:

- The publication must be published in the last twenty years (2002-2022).
- The publication is written in English.

¹<https://www.ub.tum.de/tumopac>, accessed August 2022

After selecting the literature sources, we proceeded to define the search terms used. We decided on six different search terms, which were formulated based on prior experience and other publications in the field of security metrics [47, p.145]. Some search strings are security-specific, while others refer to general software metrics to not exclude metrics listed in publications which describe metrics of different types. A list of the search strings can be seen in Table 3.1.

ID	String	Category
SSEM	Secure AND Software AND Engineering AND Metrics	Security
ITSM	IT AND Security AND Metrics	Security
SSM	Software AND Security AND Metrics	Security
DSOM	DevSecOps AND Metrics	Security
SDM	Software AND Development AND Metrics	General
SQM	Software AND Quality AND Metrics	General

Table 3.1.: The defined search strings including their abbreviation and category

Phase 2: Synthesis and analysis

With the selected literature search engines and the defined search strings we conducted our structured literature review. During the review it became apparent, that some used literature search engines delivered many results and therefore a stopping criterion was required for the feasibility of the research. For these search engines the stopping rule described by Garousi was used to abort the search [46, p.17]. It states, that the first 100 search hits are always examined and afterwards the search only continues, if there are relevant hits that revealed additional search results on the last page of the search engine. Using these termination criteria, you can see the number of relevant unique publications identified by each source and search term in Table 3.2. In the total the research includes 77 unique publications from the seven used databases and search engines. All search strings used provided new results.

		Security				General		Sum
		SSEM	ITSM	SSM	DSOM	SDM	SQM	
	IEEE	3	6	5		2	3	19
①	ACM	1		1		1		3
	Science Direct				1			1
	Scopus	1	1	3	1	4	1	11
②	Google Scholar		1	4	5	1	3	14
	OPAC		5	2		2	1	10
③	Grey Literature		3	5	11			19
	Sum	5	16	20	18	10	8	77

Table 3.2.: The amount of relevant publications found in the defined literature sources

As described in Section 2.3.1 and 2.3.2, metrics can be of variable quality. This became evident within the described security metrics in the identified literature. We therefore carried out a metric-specific procedure so only high-quality security metrics qualify for the catalogue. It consists of three stages and only if a metric passes all three stages of this procedure it is included in the catalogue.

1. Assuring that the candidate metric specifically measure security aspects
2. Applying the qualification criteria of Jaquith introduced in Section 2.3.1 to the metric [58]. To be included in the catalogue, the candidate needs to be
 - consistently measured without subjectivity
 - cheap to gather
 - expressed as a cardinal number or percentage
 - expressed using at least one unit of measure
 - contextually specific
3. Calculating the SMMM score of the candidate introduced in Section 2.3.2 [85]. To be included, the metrics needs to receive a score of at least 85%, which indicates that the metric is of the highest maturity according to the SMMM specification.

Table 3.3 lists the candidate metrics that did not qualify for the catalogue, along with their source and the reason for not being included. The disqualification reason is presented with the incomplete phase of the procedure (the second as ② and the third as ③) and the specific cause within the phase. In the overview we disregard all metrics found in the analyzed literature that did not pass the first phase of the procedure, e.g. that do not measure security aspects but other SDLC activities. In total, twelve security metrics that were proposed in the literature did not qualify for the catalogue.

Metric Name	Source	Reason
Number of security requirements	[11], [131], [6], [120], [56]	② Contextually unspecific
Ratio of security requirements	[131], [6], [120]	③ SMMM score of 77%
Number of threats identified	[11]	② Contextually unspecific
Number of relevant attack patterns	[11]	② Contextually unspecific
Number of high risk statements	[11]	② Contextually unspecific
Alerts created	[8]	③ SMMM score of 69%
Reduction in security related tickets	[142]	③ SMMM score of 58%
Total security test duration	[14]	② Contextually unspecific
Number of used security algorithms	[131]	③ SMMM score of 66%
Number of security design flaws	[131], [6], [120]	③ SMMM score of 64%
Stall ratio	[30], [6], [120]	③ SMMM score of 77%
Security requirements stage errors	[6], [56]	③ SMMM score of 75%

Table 3.3.: The list of unqualified candidate metrics

3.2. Interviews

To evaluate the representation of a team's security maturity in our proposed tool (see Chapter 6), we conducted interviews with various stakeholders in the agile development process.

Study design

The goal of the interviews was to obtain feedback and opinions specifically on the following areas:

1. Visualization of the security maturity score and its components
2. Input of the necessary information such as assessments and security metrics
3. Comparison between the proposed security maturity approach and other security approaches used in practice

We chose to interview software engineers and consultants who work in agile development teams, as they make up an agile development team and have a primary interest in security metrics. In addition, we conducted an interview with a security specialist who uses security tools and security metrics in practice and is therefore qualified to provide his feedback on the third objective. A semi-structured questionnaire was used for the interviews, which can be found in the Appendix A. We chose this approach as it allows for individual adjustments depending on the role or experience of the interviewee or the course of the interview [63, p.4]. The interviews were conducted synchronously via a videoconferencing tool and recorded to allow for transcription, following the ACM standard for qualitative surveys [121]. Firstly, the interviewer introduced the research space and the to be evaluated concepts. Afterwards, the features of the prototype were discussed by presenting it asking open-ended questions to gather information from the participants. With this structure, we achieved the evaluation objectives and the interviews consistently adhered to the ACM standard.

Data collection

In total, we conducted eight interviews in the described format, each lasting between around half an hour to 45 minutes. An anonymized overview over the participants, their company role and the duration of the interviews can be found in Table 3.4. The results of the evaluation are described in Section 7.4. After the penultimate and final interview, we received little new information and therefore concluded the validation.

Abbr.	Role	Duration (mm:ss)
P1	Software Consultant	44:21
P2	Software Engineer	40:05
P3	Software Consultant	35:40
P4	Software Engineer	42:34
P5	Software Engineer	40:46
P6	Principal Engineer	37:17
P7	Software Consultant	32:37
P8	Software Engineer	25:32

Table 3.4.: The participants of the conducted interviews

4. Related Work

In this chapter we discuss previous work in the area of security metrics and team security maturity that forms the basis for the research. We also describe where the research leverages the accomplishments of the work, extends previously proposed concepts, or departs from the results obtained. The order of the sections follows the proposed research questions (see Section 1.2).

4.1. Security metrics catalogue

Various research groups have made great efforts to propose and validate different collections of software metrics of different types, such as performance, code quality, or security. Together they form the group of publications analyzed in the literature review (see Section 3.1). Some of these publications contain a catalogue of security metrics. Three of these publications and their catalogues are presented in the following sections.

Catalogue of metrics

The research provides a collection of security metrics structured according to the different phases [131]. It has been prepared in accordance with the Goal-Question-Metric approach, which first identifies the appropriate measurable goals for achieving software security and then selects the appropriate metrics to measure the achievement of the goals [124], [131, p.461]. In this approach, the proposed security metrics are not the result of a structured literature review, but are new suggestions, which introduces a validity problem. Moreover, the catalogue does not include metrics that are not part of a single phase of SDLC, as in our case, where some metrics also have an overarching character across all phases.

Software quality metrics

The work describes 19 quality metrics and applies them in a case study of more than 3,000 engineers on various teams, 20 percent of whom use agile methods [102]. The metrics collection itself comes from Cisco Systems¹, which has fully implemented it across its on-premise products. The catalogue is divided into four so-called enforcement points, which are prevention, inspection, evaluation and remediation, and are not the same as SDLC. The results of the case study show that the introduction of the quality metrics led to, among other things, a reduction in the time to reach software maturity and a reduction in the annual defect

¹<https://www.cisco.com>, accessed November 2022

rate. The collection includes some security-specific metrics and some general software quality metrics, while our catalogue focuses exclusively on security issues. It also does not provide guidance on how to measure the metrics (e.g., to what level of automation or with what tool), and it lacks the linkage of the enforcement points to the individual phases that our catalogue contains.

KPIs for evaluation of DevOps teams

In this paper, 14 metrics of different types are presented and described as a result of a structured literature review. They are classified into four types: delivery, cost, defects, and tests [47]. In addition to the types, the authors describe some suggestions on how to measure the metrics and explain the implications of an unsatisfactory result of a metric. Similar to the Pradhan et al. publication, this collection neglects some critical metric properties that we include in our catalogue, such as the degree of automation or tool support of a metric. In addition, the catalogue does not link metrics to SDLC phases and does not include reference values.

4.2. Security metrics for team security maturity

In our literature review we were able to identify several models that use software metrics to calculate a team maturity score, which we present in the following sections.

TSM

The TSM is a maturity model specifically assessing the security capabilities of a development team during the SDLC [144]. It consists of three pillars “self-assessments”, “external assessments” and “automated”, which represent the three different information sources used by the model. Each pillar includes multiple security domains such as “Knowledge” or “Culture”, which themselves consist of multiple security topics. As the name suggests the first two pillars use assessments in the form of questionnaires, which an internal team member or external auditor fills out to assess single topics. Each topic can be assessed in four different maturity levels, namely “no”, “partly”, “largely” and “fully”. The third pillar “automated” includes a preliminary suggestion on converting the results from different security tools into the four maturity levels.

While the third pillar “automated” is a helpful starting point for organizations, it doesn’t include any KPIs or security metrics that are suitable for the maturity calculation. Additionally, it doesn’t include any information on how to convert the results from the security KPIs to the four maturity levels. With our approach we address both of these issues.

SMMM

The SMMM and its five maturity levels can help an organization understand how mature its security metrics are [85]. The maturity level of a metric can be calculated by rating different attributes of a metric with different qualitative and quantitative properties. The maturity value of a metric is represented as a percentage between 0 and 100 and depending on the score has one of three maturity levels, namely infant, evolving or matured. Once each metric is evaluated, an overall metric maturity score in the organization can be deduced.

Firstly, the SMMM assesses the maturity of security metrics, whereas we explore how to assess the security capabilities of a development team. In addition, the model does not include an approach to assess the current performance of the metrics, which our model focuses on. Finally, the model yields an overall maturity score at the organizational level, while our approach focuses on assessing development teams.

5. Security metrics

This chapter describes our security metric catalogue, which is the first research artifact and answers RQ1. The catalogue consists of qualified security metrics, containing a description of each security metric in the structure proposed by Bouwers et al. (see Section 5.1). It is the result of an extensive systematic literature search and a rigorous qualification procedure described in Chapter 3. In total, it contains 31 metrics which describe team and product security attributes. The full catalogue can be found in the Appendix B.

5.1. Catalogue structure

The catalogue is structured in a two-level hierarchy. First, we divide the selection of security metrics into two groups, team security metrics and product security metrics. This follows the accepted definition for security metrics for our research (see Section 2.3) and the commonly accepted property of security metrics in the literature, which states that a metric can measure either the capabilities of a team or the properties of a developed artifact (e.g. [81, p.1]). At the second hierarchical level, security metrics are grouped into the domains of the SDLC, again following the structure of the accepted definition of SDLC in Chapter 2.

Each metric is described with several attributes that describe either a qualitative or a quantitative property of the metric. This structure and all attributes come from a publication by Bouwers et al. that describes a catalogue format for software metrics of all categories and is therefore naturally also suitable for use with security metrics [20]. The format consists of 16 qualitative and quantitative attributes. The qualitative attributes describe what insights a metric can provide, how it is measured, and what actions can be taken to improve the measurement. On the other hand, the quantitative attributes describe the expected value of the metric, the possible range of measurements, and whether the metric is an absolute value or a ratio. In addition to these attributes, we introduce two additional fields that describe the extent to which the metric can be measured automatically and the type of tool that can be used, to answer the second part of RQ1. The automation attribute is described in three different ordinal degrees of automation: ○ manual, ● semi-automated, and ● automated. We define manual metrics as metrics that cannot currently be measured automatically by any tool or that require an individual measurement component. Measurement of semi-automated metrics relies on one manual and one automated component, so for a ratio metric, the numerator must be measured manually, but the denominator can be measured automatically. Absolute metrics can be semi-automated if their measurement is tool-based but requires some manual effort, such as entering a value. Automated metrics can be reliably measured without a manual factor using existing tools. The tool property can include any security tool type,

such as SAST or DAST, described in Chapter 2.

In total, we propose a catalogue of 31 security metrics, divided into six team security metrics and 25 product security metrics. 15 security metrics, or almost 50%, can currently already be measured fully automatically, while only six metrics, or about 20%, are classified as manual. Figure 5.1 shows this and some other insightful distributions of the metrics attributes.

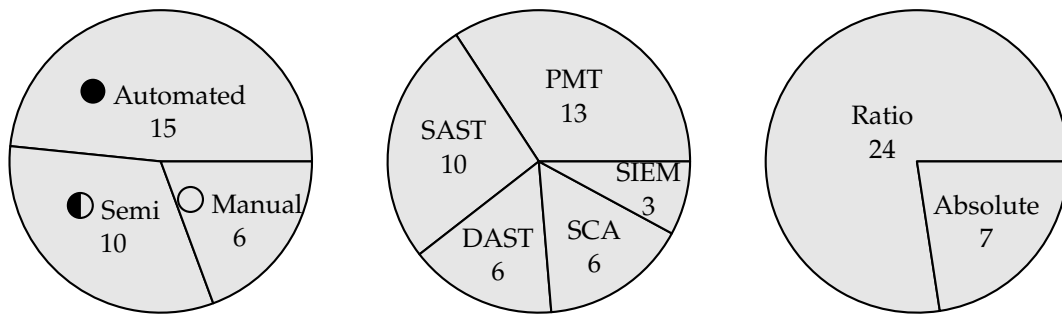


Figure 5.1.: Distribution of different attributes amongst the identified security metrics

We found several advantages of the chosen catalogue structure. First, the catalogue provides a bidirectional information path: On the one hand, the group of security metrics relevant to a particular phase in the SDLC can be easily selected. On the other hand, it is easy to trace in which phase of the SDLC a particular security metrics is applied. Moreover, it is easy to find out which entities are measured during the SDLC and which need to be measured in other ways. Currently, a total of 18 different entities are measured during all phases, e.g., dependencies, security requirements, or security audits. Finally, the field of related metrics makes it easy to understand possible correlations between the security metrics.

5.2. Team security metrics

In the following two sections we describe the two domains of security metrics that include security metrics for measuring team attributes. We list the identified security metrics with their ID, name, automation level, and tool category. In addition, the SMMM-score calculated using the definition introduced in Section 2.3.2 is included. Listed tool categories marked with an asterisk (*) are theoretically described by the metric validation sources, but the example tool in the category presented in Section 2.3.3 cannot measure the metric. The full description of the metrics can be found in the Appendix B. In Appendix B.3 the complete calculation for all metrics is presented.

5.2.1. Knowledge domain

In a nutshell, knowledge security metrics measure how well the team is informed about current security practices and requirements. This follows the definition of the TSM knowledge domain in the self assessment pillar [144, p.51]. Updated knowledge and high-quality

information in a team is critical to its security performance and therefore this domain is of high importance [52, p.3]. Two security metrics are described in the reviewed literature.

ID	Name	Automation	Tool	Score
CSPPR	Company Security Policy Review Rate	● Automated	PMT	100%
GSRRR	Government Security Regulation Review Rate	● Automated	PMT	100%

Table 5.1.: Security metrics in the knowledge domain

The reason for the limited selection could lie in the difficulty of measuring a team’s tacit knowledge, which is implicit knowledge held in the team member’s minds and is difficult to transfer to another person or make explicit [113, p.1614], [114, p.229]. The collected security metrics in this domain take account for this difficulty by acting as proxy metrics, which substitute the direct measurement of tacit knowledge by measurements of the surrounding properties of the team [114, p.232]. Additionally, the security metrics of the “Effort” domain complement the selection.

5.2.2. Effort domain

Building on the first team domain “Knowledge”, metrics in the effort domain try to measure, how much time a team collectively spends productively on security efforts during the SDLC. Such security effort activities can include trainings or special security focused team sessions [108, p.2]. We identified four security metrics in this domain.

ID	Name	Automation	Tool	Score
SAER	Security Awareness Effort Rate	● Automated	VMT	88%
SRER	Security Remediation Effort Rate	● Automated	SAST	88%
SMC	Security Meetings Count	◐ Semi-automated	PMT	85%
SAPR	Security Audit Pass Rate	○ Manual	PMT*	94%

Table 5.2.: Security metrics in the effort domain

The concept of tacit knowledge introduced in Section 5.2.2 above is closely related to skill learning and communication [114, p.230, p.238]. Therefore, we regard measuring the awareness efforts and the communication in a team is highly important and relevant to a team’s security level, which is possible with the proposed metrics in this domain.

5.3. Product security metrics

In the following five sections we describe the five domains of product specific security metrics. Product specific metrics are intended to be measured for every product a team develops individually. This way the processes for individual products can be reviewed while also being able to get an overview of a team's general performance in a SDLC phase across all products. Again, we list the identified security metrics with their ID, name, level of automation and tool category and the full description then can be found in the Appendix B. In addition, the SMMM score calculated is included, which components can be found in Appendix B.3. Equivalent to the security metrics team, for the tool categories marked with an asterisk (*) our exemplary tool within that category can not measure the metric.

5.3.1. Analysis Domain

Security analysis in requirements engineering is an important activity of the first SDLC phase. The required security level and security challenges of the developed product should become apparent, discussed and recorded, so the team understands what lies ahead in the development process. Without extensive analysis efforts, all subsequent phases can be deficient. Therefore, it is important that already during this phase security metrics are measured and reported. In total the literature includes three metrics to measure the thoroughness of this phase.

ID	Name	Automation	Tool	Score
OSRC	Omitted Security Requirements Count	○ Manual	PMT	92%
SRGAR	Security Requirements General Analysis Rate	● Semi-automated	PMT	85%
SRTMR	Security Requirements Threat Modeling Rate	● Semi-automated	PMT	100%

Table 5.3.: Security metrics in the analysis domain

The small amount of found security metrics that measure the analysis efforts can be explained with the intangible nature of security requirements engineering. While it produces documented records, the process is carried out in heterogeneous, irreproducible steps and even the final requirement records are written only in semi-technical, human-readable documents. This is also reflected by the fact that all security metrics are only trackable by PMTs.

5.3.2. Design Domain

System design may be the most important SDLC phase of the shift left security trend (see Section 2.1), as in this phase decisions can be taken to completely avoid many - some findings show up to 50% - of the common security weaknesses [80, p.1164]. "Design" has a proportional connection with the prior phase "analysis", as all captured requirements should or rather must be addressed in system design, while omitted requirements cannot be focused on. This can lead to the conclusion that the design can only be as good as the analysis was thoroughly. It is therefore important not only to measure the quality of the

proposed architecture components, but additionally measure the connection to prior SDLC phase, e.g. with the help of the SRSACR. In total, we identified five security metrics, which measure the activities during the design phase.

ID	Name	Automation	Tool	Score
ACASAR	Architecture Component Attack Surface Analysis Rate	○ Manual	PMT	94%
ACARAR	Architecture Component Architectural Risk Analysis Rate	○ Manual	PMT	94%
SRSACR	Security Requirements Satisfying Architecture Components Rate	○ Manual	PMT	92%
CCR	Coupling Corruption Propagation	● Automated	SAST*, DAST*	85%
CER	Critical Element Ratio	◐ Semi-automated	SAST*	85%

Table 5.4.: Security metrics in the design domain

During the design phase a multitude of approaches and support tools are used to visualize the system design. In a PMT, the information of all these sources are collected and therefore it is the single source for the security metrics of this phase.

5.3.3. Implementation Domain

When met with a poor development process, even the best system design only increases security marginally [80, p.1167]. Therefore, it is critical that security metrics are also measured during implementation to verify that the system design has been implemented correctly and all implemented components are of satisfactory security. The analysed literature included six security metrics that measure code structure and test properties.

ID	Name	Automation	Tool	Score
STCR	Security Test Coverage Rate	◐ Semi-automated	SAST	92%
STSR	Security Test Success Rate	◐ Semi-automated	SAST	92%
STFR	Security Test Failure Rate	◐ Semi-automated	SAST	92%
DSVC	Dependency Security Vulnerabilities Count	◐ Semi-automated	IAST, SCA, VMT	94%
DSVR	Dependency Security Vulnerabilities Rate	● Automated	SCA	94%
UER	Unsecured Endpoints Rate	● Automated	SAST, DAST, IAST, VMT	100%

Table 5.5.: Security metrics in the implementation domain

The implementation phase of the SDLC is the first phase, where all metrics can be measured at least semi-automatically with the help of security tools, as code is of tangible nature and can be automatically analysed.

5.3.4. Deployment Domain

The deployment is the step in the SDLC where a software artifact becomes available and all security weaknesses potentially exploitable. Therefore, it is vital to measure security metrics at this final stage before an artifact is deployed or goes into production. This importance is reflected in the literature, as it describes more security metrics in the deployment domain as in any other SDLC phase.

ID	Name	Automation	Tool	Score
VSCR	Vulnerability Scanning Coverage Rate	● Automated	SAST*, DAST*, SCA*	100%
CSDR	Code Scanning Detection Rate	● Automated	SAST*, DAST*, SCA*	100%
OVC	Open Vulnerabilities Count	● Automated	SAST, DAST, IAST, VMT, SIEM	92%
OVR	Open Vulnerabilities Rate	◐ Semi-automated	SAST*, DAST*, SCA*	94%
MVC	Mitigated Vulnerabilities Count	● Automated	SAST, VMT	92%
VSR	Vulnerability Slippage Rate	● Automated	SIEM*	88%
CFR	Change Failure Rate	● Automated	PMT	100%
SDC	Spillover Vulnerability Count	● Automated	PMT*	92%

Table 5.6.: Security metrics in the deployment domain

Similar to the implementation phase described in Section 5.3.3, during deployment all security metrics can at least be measured semi-automatically - in fact all measurements except for the OVR can be carried out automatically. But in contrast to implementation security metrics, in this phase a multitude of tools are required for the measurements.

5.3.5. Maintenance Domain

Even when the software is believed to be secure, it is possible that new vulnerabilities or exploits become known. Metrics in the maintenance domain therefore measure how resilient a product is during runtime. Furthermore, metrics in this domain measure how quickly the team can fix a new issue. We identified three security metrics in this domain.

5. Security metrics

ID	Name	Automation	Tool	Score
AR	Availability Rate	● Automated	SIEM	100%
MTTR	Mean Time to Resolve	◐ Semi-automated	PMT	94%
MTTF	Mean Time to Fail	○ Manual	SIEM*	85%

Table 5.7.: Security metrics in the maintenance domain

In practice, implementing measuring the maintenance metrics can be challenging. While the AR is often reported by SIEM tools automatically and can be measured easily, MTTR depends on some measurement of how long actual work was carried out [142, p.23]. Possible units include hours, days, man-hours, or full-time equivalents and depending on the choice a conversion may be needed and the calculation is changed. Without any SIEM tool, the total production time needs to be tracked separately, e.g. in hours or days.

5.4. Measuring security metrics with security tools

In addition to the automation attribute in the metric catalogue we investigated which tools are able to measure and report the identified metrics.

Firstly we selected one exemplary tool, which is either open source or well documented and widely available, for each of the in Section 2.3.3 described tool categories. Then we analysed each tool by reading the provided documentation, if possible installing a test instance of the tool and running it on a project called OWASP WebGoat¹, which is a deliberately insecure web application for testing purposes. Figure 5.8 describes which of the chosen exemplary tools can collect which of the collected metrics.

With this overview it becomes apparent that eight of the identified metrics cannot currently be collected with any of the analysed tools, which is roughly a quarter of all proposed security metrics. This lack of automated measurement hinders the implementation of the catalogue and acts as an inhibitor for security metrics applications [140, p.95]. However, as these metrics are described in the literature, a need for their implementation becomes clear. Therefore, many opportunities for new, innovative security tools or possible extensions of existing security tools exist. Additionally, it becomes clear from the overview, that some metrics such as the “Open Vulnerabilities Count” or “Unsecure Endpoint Rate” are well covered and can be collected by tools from different categories, while other metrics can only be measured by tools of a single category.

The presented procedure could also be carried out for organizations before introducing a security metrics program by analyzing its existing security tools. Depending on the results of the analysis, the organization may then need to adjust its security tools.

¹<https://owasp.org/www-project-webgoat/>, accessed September 2022

5. Security metrics

		PMT	SAST	DAST	IAST	SCA	VMT	SIEM	Sum
		Jira [9]	SecurityRAT [116]	SonarQube [125]	OWASP ZAP [96]	Seeker [133]	Dep. Track [37]	Defect Dojo [35]	OSSIM [95]
Team metrics	Knowledge	CSPPR	✓						1
		GSRRR	✓						1
	Effort	SAER						✓	1
		SRER			✓				1
		SMC	✓						1
		SAPR							0
	Analysis	OSRC		✓					1
		SRGAR		✓					1
		SRTMR		✓					1
	Design	ACASAR		✓					1
ACARAR			✓					1	
SRSACR			✓					1	
CCR								0	
CER								0	
Product metrics	Implementation	STCR		✓					1
		STSR		✓					1
		STFR		✓					1
	DSVC				✓	✓	✓	3	
	DSVR					✓	✓	1	
	UER			✓	✓	✓	✓	4	
Deployment	VSCR								0
	CSDR								0
	OVC			✓	✓	✓	✓	✓	5
	OVR						✓		0
	MVC			✓			✓		2
	VSR								0
	CFR	✓							2
	SDC								0
Maintenance	AR							✓	1
	MTTR	✓							1
	MTTF								0

Table 5.8.: The measurement of security metrics in exemplary tools

6. Team security maturity through security metrics

In this chapter, we describe how security metrics can be used to compute a team security maturity score. First, we integrated all security metrics presented in Chapter 5 into the “Automated” pillar of the TSMM and set four maturity levels for each metric. Then using the pillar we propose a way to calculate an overall security maturity score across all pillars and dimension.

6.1. Security metrics in TSMM

To integrate the security metrics from the catalogue into the TSMM a conversion from the catalogue structure to the TSMM structure is necessary. Firstly, we assign each metric a TSMM specific identifier, which follows the the TSMM naming scheme. Secondly, four attributes for each metric are selected: the name, the domain, a short description and the related metrics. We decided for only these four attributes, to be coherent with the structure of the other two pillars. Even though the other catalogue attributes are emitted, they can be attached to the TSMM to be a helpful resource while evaluating the security maturity score. Especially information about the automation level and potential tool support of the metric should be attached, as this could support the choice of tools by a user of the TSMM.

In addition to the attributes, each metric has four value ranges for the four maturity levels of the TSMM, which are “No”, “Partly”, “Largely” and “Fully”. A value range has an upper and lower bound and a measurement within the bounds is assigned the matching maturity level. The ranges and their allocation to the levels depend on the scale type, the expected value, and the range of the metric described in the catalogue. Jointly, they must also be complete and exclusive, so that each measurement always has exactly one maturity level. With them any metric measurement can be converted into a maturity level: for instance a metric measurement of 72% is sorted into the maturity level “Largely”, if it has the value range >50%-75%. The current value ranges for metrics with a ratio scale type follow a strict scheme: the expected range described in the catalogue is split into for value ranges of equal size. For the other metrics the value ranges were chosen with peer-reviewed and grey literature and only act as a broad suggestions.

The value ranges can also be adapted in an organization to be applicable to more ambitious or lenient maturity approaches. For instance when the value ranges for the upper levels are shrank and for the lower levels enlarged, only very good measurements can achieve a good maturity level.

In Table 6.1 all attributes and the value ranges for an exemplary metric are described.

ID	Topic	Description	Related security metrics	
Metric-K-1	Company Security Policy Review Rate	Percentage of relevant Company Security Policies reviewed	Metric-K-2	
	No	Partly	Largely	Fully
	0%-25%	>25%-50%	>50%-75%	>75%-100%

Table 6.1.: Exemplary description of a security metric in the knowledge domain

As a majority of the integrated metrics measure product specific attributes we introduced a product specific part to the TSMM, whereas before only team specific topics were present in the model. This means that these metrics need to be measured for each product a team develops, and then aggregated together with the team security metrics to a single maturity score. This step was necessary, as only the team and product metrics together can give a holistic overview over the teams capabilities. However it also introduces a new level of complexity, which can be addressed by using the in Chapter 7 introduced TSMM prototype. In Figure 6.1, the new structure TSMM is visualized as a UML diagram.

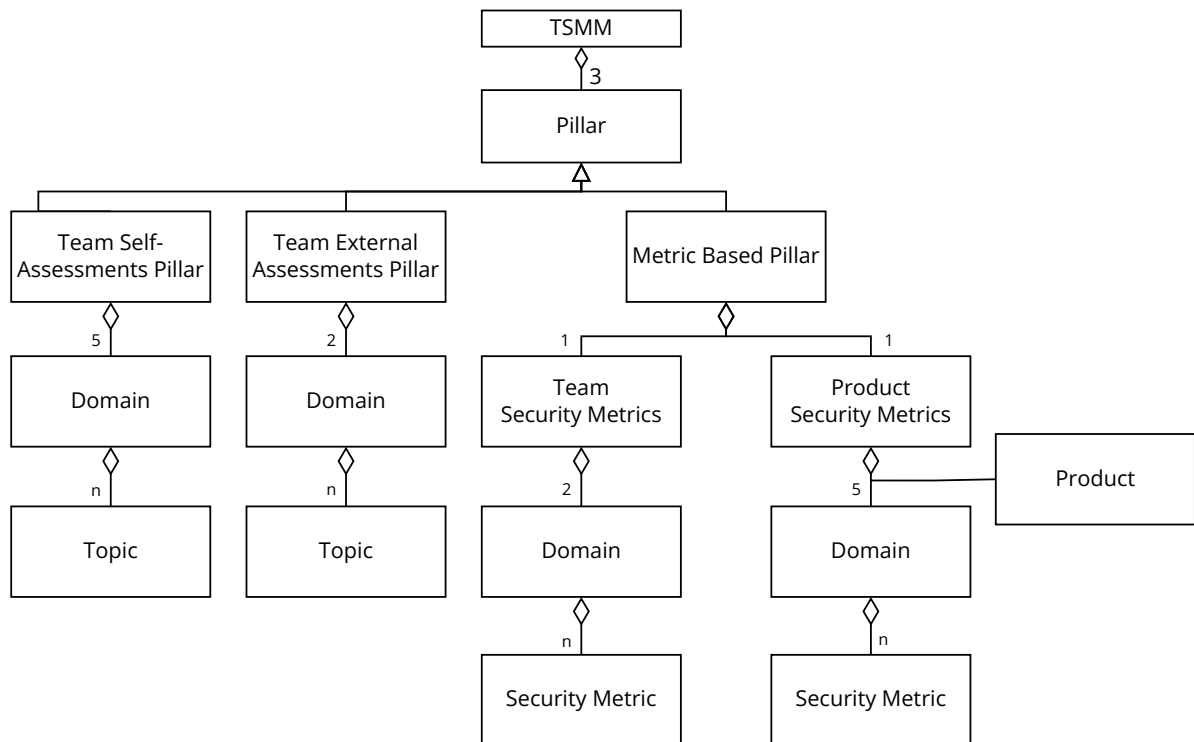


Figure 6.1.: A high-level overview over the adapted TSMM pillar and domains

6.2. Calculation of the overall security maturity score

In the initial publication of the TSMM, the model is introduced without a method to calculate an overall maturity score of the team across all pillars, domains and topics (compare [144]). We therefore introduce a new approach to calculate an overall maturity score as a normalized percentage score between 0% and 100% across all dimensions of the model. Such an overall assessment is useful because, it shows at first glance whether there is room for improvement for the team [85, p.105]. It can also act as a commonly understood and accepted language and help communication between the different stakeholders [70, p.271].

To enable the overall maturity score we assigned each of the four maturity levels a single percentage as the basis of all further calculations. The assignment we chose was No - 0%, Partly - 33%, Largely - 66% and Fully - 100%, to achieve an even distribution of the four different levels. The calculation of the overall score then requires the following steps in each of the three TSMM pillars:

- **Team Self-Assessment pillar:** (analogously the Team External Assessment pillar)
 1. Assign each assessed topic the maturity level value - if a topic is not assessed, it is disregarded in the calculation
 2. For each domain, calculate the arithmetic mean of all its topics maturity scores, to receive the domain maturity scores
 3. Calculate the arithmetic mean over all domain maturity scores to receive the maturity score of the pillar
 - **Metric-based pillar:**
 1. For the two team security metrics domains, assign the matching maturity level to the latest metric measurement value by using the introduced value levels - if a metric has not been measured at all, it is disregarded in the calculation
 2. For each of these domains, calculate the domain's maturity score by calculating the arithmetic mean of the topics maturity scores, to receive the team domain's maturity scores
 3. Now for each product separately assign the latest measurements for all product security metrics to their maturity level - again if a product metric has not been measured yet, it is disregarded in the calculation
 4. For each of the product security metrics domains, calculate the arithmetic mean of its topics maturity scores
 5. For all products calculate the arithmetic mean of both the team and product domains to receive the products maturity scores
 6. Calculate the arithmetic mean over all product maturity scores to receive the pillar maturity score
- Finally calculate an arithmetic mean over the security maturity scores of the pillar and round it to the nearest whole number to receive an overall security maturity score

Alternatively other approaches to calculate the domain maturity score in the pillars exist, which punish or reward outlier performances of one of the topics in a domain. On the one hand it is possible to replace the arithmetic mean with a minimum operation, where the lowest topic score in a domain is the overall maturity score of the domain. This very strict calculation leads to worse security maturity scores and could be applied in teams, where security is of utmost priority. On the other hand to reward an outstanding performance in one topic it is possible to replace the arithmetic mean with a maximum operation, where the best topic score is used as the domain score. This approach could be taken, when security is regarded as low priority. For both alternative approaches the calculation should be clearly communicated and made understood in the teams. For our research we use the default approach using the arithmetic mean.

To better visualize the approach in the following we calculate the security maturity score with an example. The calculation starts after the security maturity levels were assigned to the metric measurements in the metric based pillar and the domain maturity score for all domains was calculated. In Table 6.2 all domain maturity scores are listed.

Self Assessments	Knowledge	Documentation	Activities	Build & Deployment	Organization
	76%	59%	72%	89%	93%

External Assessments	Audit	Culture
	81%	86%

Metric-based	Product	Knowledge	Effort	Analysis	Design	Implementation	Deployment	Maintenance
A	78%	64%	64%	77%	81%	71%	90%	76%
B	78%	64%	64%	56%	43%	70%	85%	90%
C	78%	64%	64%	67%	65%	74%	80%	76%

Table 6.2.: Exemplary maturity scores of the domains in the three TSMM pillars

With these exemplary values the calculation of the overall maturity score can start. First the arithmetic mean of the maturity scores for the self- and external assessment pillars are calculated:

$$\text{Self Assessment Maturity Score} = \frac{76\% + 59\% + 72\% + 89\% + 93\%}{5} = 77.80\%$$

$$\text{External Assessment Maturity Score} = \frac{81\% + 86\%}{2} = 83.50\%$$

For the metric based pillar, firstly the arithmetic mean for each product is calculated. To receive the maturity score of the pillar the arithmetic mean over all product scores is calculated.

$$\text{Product A Maturity Score} = \frac{78\% + 64\% + 77\% + 81\% + 71\% + 90\% + 76\%}{7} = 76.71\%$$

$$\text{Product B Maturity Score} = \frac{78\% + 64\% + 56\% + 43\% + 70\% + 85\% + 90\%}{7} = 69.42\%$$

$$\text{Product C Maturity Score} = \frac{78\% + 64\% + 67\% + 65\% + 74\% + 80\% + 76\%}{7} = 72.00\%$$

$$\text{Metric based Maturity Score} = \frac{76.71\% + 69.42\% + 72.00\%}{3} = 72.71\%$$

Finally, we can calculate the overall maturity score by calculating another arithmetic mean across all three pillars and rounding it to the nearest whole number.

$$\text{Overall Maturity Score} = \left\lfloor \frac{77.80\% + 83.50\% + 72.71\%}{3} \right\rfloor = 78\%$$

Based on the example calculation, it is clear that the approach provides insights that are not visible without calculating the domain, pillar and overall maturity scores.

First, the approach allows one to see the security level of each domain in a pillar. These maturity scores of the individual domains can be used to understand the reasons for the overall maturity score. More importantly, they can provide an indication of satisfactory or inadequate team security efforts in a particular domain and point directly to areas of focus. Second, the metrics-based pillar makes it clear which products have a satisfactory or insufficient level of security. A team can use this information to focus its efforts on less secure products. In addition, the domain maturity scores in the metrics-based column indicate how well the team is performing in the various phases across all products.

Finally, domains from different pillars can be compared, such as the “Knowledge” domain in the self-assessment and the metrics-based pillar. A large discrepancy between the two maturity levels could indicate that there is a divergence in the team between the perceived level of security and the actual level of security achieved.

7. Tool-supported security maturity

This chapter describes in conceptual and technical detail the response to RQ3 its reasoning. We propose a tool that demonstrates an approach that holistically implements all aspects of the TSMM: the assessment of the self- and external assessment pillars, the automated collection of security metrics by tethered security tools, the calculation of a team's overall maturity level, and finally the visualization of the TSMM results. The tool is based on a formerly developed application called principle self-assessment tool for agile development teams (Prince).

7.1. Use cases

User-centric software engineering (USE) describes a collection of activities carried out during the SDLC to enable the delivery of systems that meet the user's needs [34, p.34]. We adhered loosely to USE during development of the frontend of the prototype, as it often can be advantageous to create high-quality software artifacts [34, p.35]. As the framework involves many activities but the proposed software artifact was prototyped from the beginning, we performed a subset of two USE activities. The first activity we performed is the identification of user groups and the subsequent creation of typical user stories for the different groups. We identified two potential user groups for the developed artifact with different information requirements: the developers on the team and external security specialists.

Developers. Developers typically have a strong interest in the security level of their applications and what needs to be done to improve it [123, p.248]. To address this information need, we created two user stories. In our prototype, developers should be able to see the current level of various security domains, and they should be able to examine which metrics contribute to that level. In this way, developers receive information about which measurements are satisfactory and which need to be improved. They should also get guidance from our tool on how to improve low performing domains.

Security specialists. As introduced in Section 1.1 security specialists are experts in a central security department who oversee the company's efforts to develop secure software. They analyze the current security status of existing applications, help development teams resolve security issues, and report to supervisors and managers on the company's security posture. We subsequently created two user stories in the context of these activities. Firstly, security specialists should be able to see measurements of all security metrics on a team or application and discover security domains that need improvement. Secondly, experts should be able

to use the security maturity score and all measurements to report the security status of an application or some aggregation of multiple applications to other stakeholders.

During development, we followed the four presented user stories of the two identified user groups, and our proposed artifact includes functionality for all of them (see the 7.3 section). All other implemented and proposed features do not fulfill any information requirement, but serve as enablers for these main user stories and are therefore not captured with user stories.

The second USE activity we performed was an evaluation of the developed prototype with members of the two user groups to get feedback on the implemented features and the user interface. The extensive results of the evaluation are presented in Section 7.4.

7.2. Backend application

The overarching goal for customizing the backend application is to introduce the maturity calculation described in section 6.2. According to our proposed customized version of TSMM, the maturity calculation consists of three components, namely self-assessments, external assessments, and security metrics. Only the security metrics component and the maturity level calculation itself needed to be implemented, as Prince already supported manual assessments. We describe these two parts in the next two sections.

7.2.1. Security metrics collection

As described in Chapter 5, most metrics can be collected automatically or at least semi-automatically by security tools. Therefore, to obtain security metrics in our prototype, we decided to build several REST API connections to important existing security tools. These connections retrieve measurements of specific security metrics from a security tool, then the received data is parsed into metric records in our backend, and finally the records are persistently stored in the database. Once set up, this process is triggered periodically, and at each trigger, the connections request the latest measurement of their respective metrics and create a new record for each measurement. The currently supported connections and their corresponding metrics are listed in Table 7.1. In total ten metrics are currently automatically collectable with Prince, which is just under a third of all security metrics listed in the TSMM.

Tool Category	Tool	Metrics
SAST	SonarQube	SRER, OVC, STCR, STSR, STFR
SCA	Dependency Track	DSVR, DSVC
VMT	Defect Dojo	UER

Table 7.1.: The currently supported tools in the proposed prototype

Although the choice of tools may be limited at the moment, the chosen software design

makes it easy to extend the proposed prototype with different security tools. To add a new connection to a new security tool, only two steps are required: First, the API path of the tool for security metrics measurements must be identified, and second, the parsing of the API response to the Prince metrics record data structure must be implemented. This simplicity is particularly worth mentioning since many participants in the evaluation pointed out the importance of simple and fast integrations with other security tools (see Section 7.4). Security metrics that cannot be measured automatically can be tracked with our prototype by manual input in the frontend application (see Section 7.3). In this way, we ensure that every type of security metrics is supported, and in addition, teams currently using unsupported security tools can manually track the evolution of normally automated metrics.

In Figure 7.1, the backend security metrics component is described as a UML class diagram. The centerpiece of the component is the security metrics records class, which contains all the information about a single measurement, such as the timestamp and the value of the measurement. It also contains the foreign key of the measured team, and if a product security metrics is measured, the product foreign key is also included. Each of the records is also captured for a specific metric, which includes information about the metric and maturity levels, and for a specific source instance, which are instances of generic security metrics sources with a unique URL and API key. Finally, information about which generic sources can theoretically measure which security metrics is persistently stored with a many-to-many relationship, since some metrics can be measured by different sources (see Table 5.8).

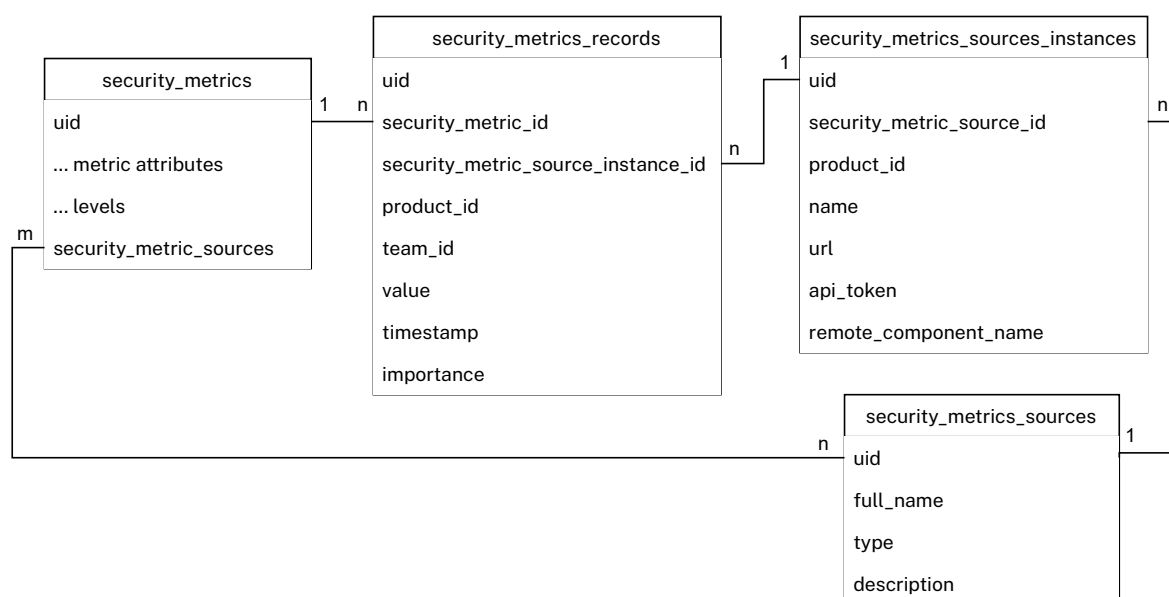


Figure 7.1.: A UML diagram visualizing the security metrics component in the proposed prototype's backend

7.2.2. Security maturity calculation

Once at least some assessments and records have been entered into the system, the calculation of security maturity for the teams can be initialized in the system. In the current implementation, the calculation is done daily by scheduling the execution with a special Spring “@Scheduled” annotation, but can be easily adapted to any other frequency or even to a manual execution. This flexibility is advantageous, as the TSMM does not include information on the frequency of the maturity calculation. For the computation, we collect the latest measurements of the security metrics and scores of all TSMM topics stored in the system. In Figure 7.2, an overview of the security maturity calculation in our prototype is visualized as a UML class diagram. With this it becomes clear that the system design follows the three-pillar structure of TSMM.

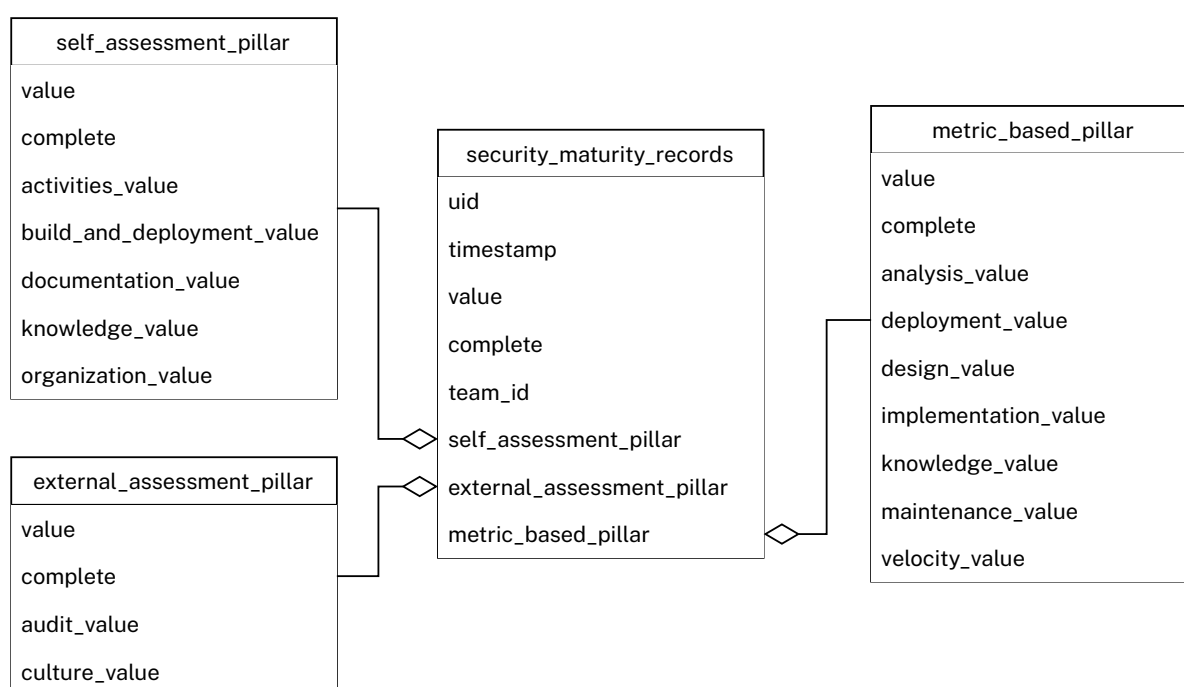


Figure 7.2.: A UML class diagram of the security maturity component in the prototype’s backend

We implemented the calculation logic exactly as described in the theory in Section 6.2: For the self-assessment and external-assessment pillars, domain maturity scores are calculated and averaged to yield a single maturity score for each domain. If a topic has never been assessed but the calculation is triggered, the topic is disregarded in the calculation, which follows the definition of the proposed calculation of an overall maturity score (see Section 6.2). For the metrics-based column, the calculation is again as described in 6.2: The first step is to assign the metric measurements to their matching maturity levels. This step is done by instantiating four maturity level objects that contain information about which values are included in their class (e.g., all values between 0-25%), and a scale class that describes whether the levels are

arranged in an ascending or descending order to enable the level assignment. If a metric has never been measured or a topic has never been assessed, it is disregarded in the calculation, which follows the definition of the proposed calculation of an overall maturity score (see Section 6.2). Then, for each product, the defined team and product metrics are averaged to obtain a product maturity score. Finally, the product maturity scores are averaged to obtain the overall metric-based column maturity score. In the listing 7.1, the maturity calculation in the metric-based column is shown. For reasons of conciseness the declaration of some variables and methods is disregarded.

After calculating the maturity score for all three columns, they only need to be averaged to obtain an overall maturity score. This overall score and additionally all other maturity scores on each level are then stored permanently and can be retrieved when reading the overall score, as can be seen in Figure 7.2. This enables to get detailed information about the calculation and the components of the maturity score, which is displayed in the frontend by the maturity dashboard (see Figure 7.7).

The implemented calculation can also easily be adapted or extended, e.g. by weighting the factors on one of the described levels. The assignment of a maturity level to a single metric measurement can also be easily adapted to one of the other calculation methods introduced in Section 6.2.

```
public class MetricBasedPillar {

    public MetricBasedPillar(Team team) {
        Double knowledge = getDomainMaturity(Domain.KNOWLEDGE);
        Double effort = getDomainMaturity(Domain.EFFORT);
        teamProducts.forEach(prod -> {
            Double analysis = getDomainMaturity(Domain.ANALYSIS, prod);
            Double design = getDomainMaturity(Domain.DESIGN, prod);
            Double implementation = getDomainMaturity(Domain.IMPLEMENTATION, prod);
            Double deployment = getDomainMaturity(Domain.DEPLOYMENT, prod);
            Double maintenance = getDomainMaturity(Domain.MAINTENANCE, prod);
            List<Double> productDomains = Arrays.asList(
                knowledge, effort, analysis, design,
                implementation, deployment, maintenance);
            Double productValue = productDomains.average();
            maturityValues.push(productValue);
        });
        maturityValue = maturityValues.average();
    }

    private Double getDomainMaturity(Domain domain) {
        List<SecurityMetric> securityMetrics = getDomainMetrics(domain);
        List<Double> maturityValues = new ArrayList<>();
        securityMetrics.forEach(securityMetric -> {
            SecurityMetricRecord record = findLatestRecord(team, securityMetric);
            if (record != null) {
                Double maturityValue = findMaturityLevel(securityMetric, record);
                maturityValues.add(maturityValue); }
        });
        return maturityValues.average();
    }

    private Double getDomainMaturity(Domain domain, Product product) {
        //...
    }
}
```

Listing 7.1: Metric Based Pillar Score Calculation

7.3. Frontend application

The developed frontend has two general goals: first, to allow the input of all necessary data for maturity calculation, and second, to make all measurements and calculated scores collected in the backend accessible. It was developed as an Angular 13¹ application written in TypeScript². To facilitate fast prototyping we additionally used Angular Material³, which solely provides user interface components with homogenous styling, and the frontend does not require any third-party dependencies.

To achieve the first goal, we first implemented a function to fill in team assessments for TSM, which is used for self- and manual assessments. This functionality was missing because previously Prince only supported product-specific manual assessments. Secondly, we implemented a form to set up connections to external security tools for collecting security metrics measurements and developed a functionality to enter manual measurements of metrics. In this way, measurements of metrics from all levels of automation can be tracked. With the described functionality, the first goal has been achieved, as all information for all three pillars of the TSM can now be entered in the frontend.

To achieve the second defined goal, we then developed a security maturity dashboard as the center of the application, following the four user stories defined in Section 7.1. It contains information about the latest maturity calculations, the evolution of the maturity level in the past, and indications of what steps are needed to improve the maturity level.

In Figure 7.3, the connection between the frontend components and the presented backend components is visualized in a UML architecture diagram. In the following sections a detailed description and a screenshot with exemplary data of the frontend features is given.

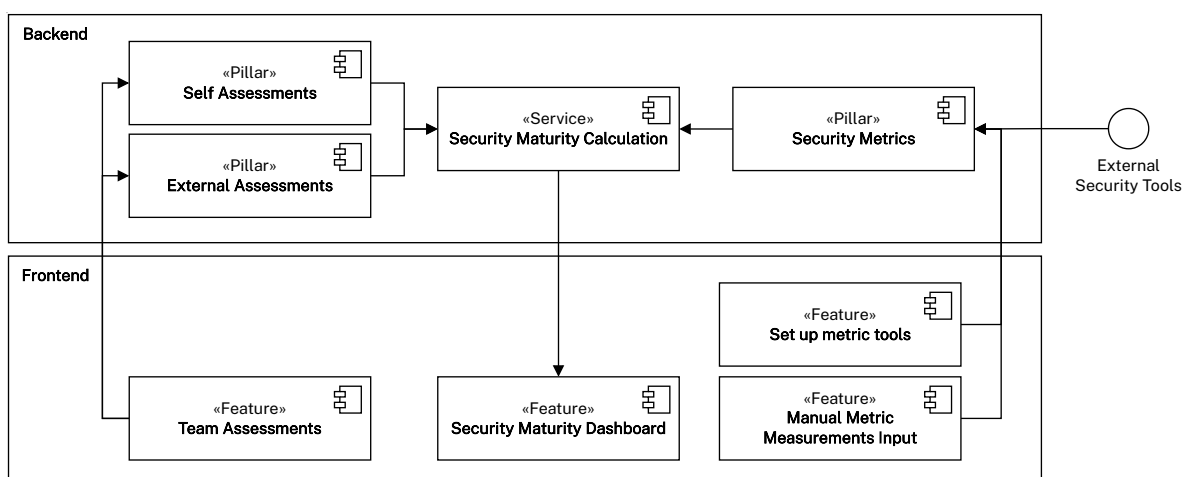


Figure 7.3.: A UML architecture diagram visualizing the connection of the frontend features to the backend components

¹<https://angular.io/>, accessed November 2022

²<https://www.typescriptlang.org/>, accessed November 2022

³<https://material.angular.io/>, accessed December 2022

Team Assessments

In Prince, the foundation for our functionality, only single products could be assessed. To enable the TSMM, we created a new feature to also allow for assessments that evaluate the whole team across all products. In Figure 7.4 a screenshot of a team self assessment is presented, where a user is currently assessing the “Build & Deployment” domain. The assessments are displayed in a tabular format, where each row represents a single assessment. On the left side the topic’s statement is listed, while on the right side the user can assess the topic with the help of a slider. When the user selects a maturity level for a topic with it, a short description of the maturity level is displayed. Additional information about the topic can be viewed by clicking on the information icon on the left, which opens a dialog with topic information and helps the user to correctly assess a topic. The information displayed includes a longer explanation of the topic, the descriptions of the different maturity levels and information about the previous assessments of the topic. Finally, after assessing some or all of the topics, the user can save the ratings persistently in the database. Then, the next time an assessment is performed, the function is initialized with the latest assessments so that the user can continue instead of starting over.

Build & Deployment













Topic	Assessment
 We have quality gates and update them regularly.	 Answer: We have some quality gates.
 We maintain all components, patch them and redeploy them on demand.	 Answer: We never patch anything.
 We maintain our backups for our components.	 Answer: We only create them for critical components. We have some processes to create them regularly, but that doesn't always work out well.
 We have a reproducible build process (with integrated security measures).	 Answer: We have a robust DevOps pipeline.
 We consider security requirements (including architecture) at all stages of the development cycle.	 Answer: Security as a requirement is constantly discussed. We tackle at all stages (development, production, ...)
 We evaluate our Build and Deploy overall as ...	 Answer: Good, but still a little room for improvement

Figure 7.4.: Filling out team assessments in the build & deployment domain

The team assessments feature is used for both self-assessments and external assessments, as both have the same process and the topics in the two pillars are structured in the same way. Nevertheless, there is a difference between the two assessments, as external assessments can only be performed by authorized security specialists according to the model specifications. Therefore, an authorization check was required so that only authorized security specialists can access the external assessments, while all team members are allowed to perform self-assessments.

Security metrics

We implemented two features to support security metrics in the frontend. First, it is possible to set up a connection to a metric source instance like a SAST tool in the frontend. This setup form can be seen in Figure 7.5. The form requires all the information required for the API connection, such as the URL and an authentication key. In addition, a remote component name is required, which is the project or product ID of the security tool, which may be different from the product name in Prince. Once this connection is created in the frontend, all inputted information is stored persistently in the database and the process of collecting measurements from the instance can begin (see Section 7.2). In addition, with the form the source instance can be edited or deleted by an authorized user if necessary.

Configure new metrics source

Name *
SonarQube Connection
Please enter a name

Tool *
SonarQube
Please enter a source

Art der Quelle *
Static Application Secu...
Please enter a source

Application *
Test App
Please enter a Product

API Token *
<api-token>
Please enter your API token

URL *
<url>
Please enter your URL

Remote Component Name *
<remote-component-name>
Please enter your remote component name

Cancel Save

Figure 7.5.: Set up of a security metrics source instance

We decided that source instances can only be created by security specialists, while the authentication key and remote component name is maintained by team members. This allocation of responsibilities should be adapted to the environment in which the tool is used.

Secondly, measurements of security metrics that cannot be collected automatically can be entered into a form in the frontend. Figure 7.6 shows a screenshot of this manual input. A user can view the metric, the corresponding application, the security tool responsible, and the old value he is updating. He can then enter new values for the metric, and once he submits the form, new metric records are created in the backend for all edits. With these two parts, measurements for all metrics of all types are represented in Prince.

Product Metrics

Abbreviation	Name	Team	Application	Tool	Value
OSRC	Omitted Security Requirements Count	Team 2	Test App	Manual	12
SRGQR	Security Requirements General Analysis Rate	Team 2	Test App	Manual	43 %
SRTMR	Security Requirements Threat Modeling Rate	Team 2	Test App	Manual	44 %
ACASAR	Architecture Component Attack Surface Analysis Rate	Team 2	Test App	Manual	55 %
ACARAR	Architecture Component Architectural Risk Analysis Rate	Team 2	Test App	Manual	80 %

Items per page: 5 1 - 5 of 25 < > >>

Cancel Save

Figure 7.6.: Manual input of product security metrics measurements

Security Maturity Dashboard

The security maturity dashboard is the centerpiece of the proposed TSMM component in the prototype. Its purpose is to fulfill all the security information requirements for the two defined user groups and to fulfill the four user stories. The dashboard is split into two parts, the first being an overview over the current maturity level and the second gives detailed information about single metrics and assessments. In the following paragraphs we present the two parts in detail.

The first part of the dashboard consists of three cards that can be seen in Figure 7.7. On the left in a gauge chart the current maturity score and the difference to the prior maturity score can be seen by a user. Above it the date of the calculation and a ranking amongst all other registered teams is visible. Below it the scores of the three pillars is visible and information is given on when the next calculation is carried out. We chose to assign each pillar its own color to be used in all dashboard visualizations, as color has been found to be one of the most powerful factors influencing a user's understanding and decision-making [132, p.18]. On the right two information cards give the current maturity score more context. On the top all domain maturity scores from the three pillars are visualized and the current maturity score is displayed as a reference line, to easily spot over- and underperforming domains. For instance in Figure 7.7 it becomes obvious, that the effort domain requires attention, as its maturity

score is below average. We decided to use a bar chart to display this information because of feedback given during the evaluation, but it is also feasible to display the information using a polar or radar chart. On the bottom the prior development of the overall and pillars maturity scores can be seen over the last ten calculations in a line chart. The sudden change in maturity level at a particular point in time could indicate a new self- or external assessment, as assessments are less frequently performed than maturity level calculations.

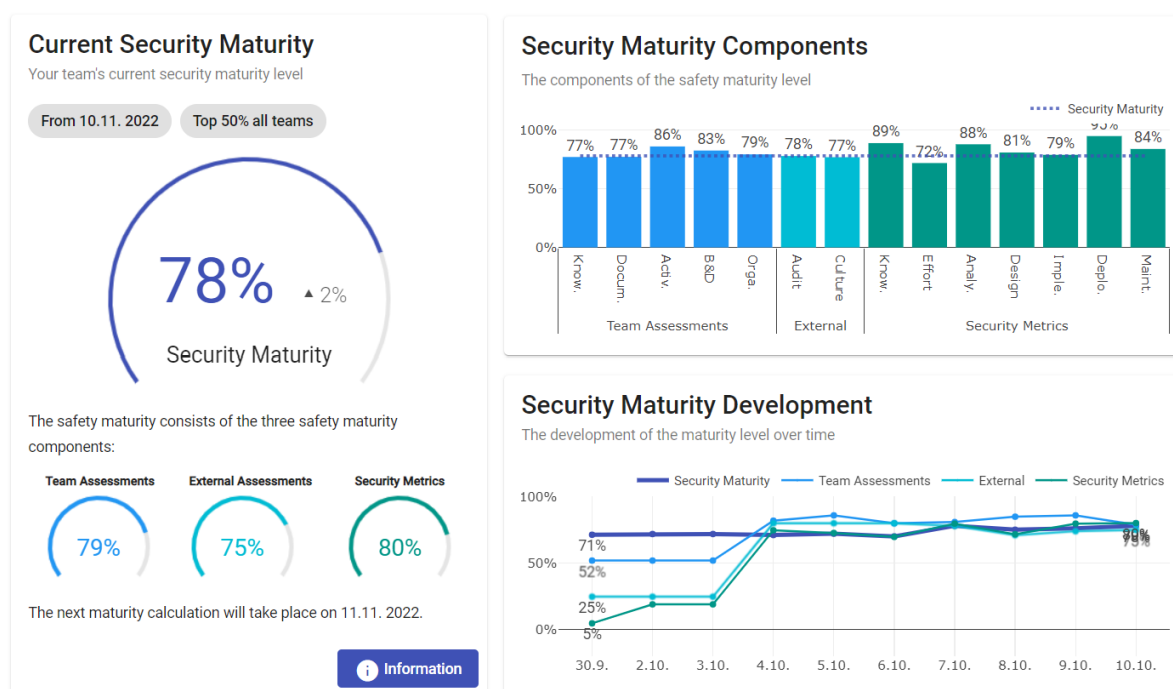


Figure 7.7.: The first part of the security maturity dashboard

In the second part of the dashboard the security metrics measurements and the assessments that made up the maturity score are presented to the user. This way the dashboard enables users to better understand what components contributed to the score.

In Figure 7.8 the overview of all the latest security metrics measurements is shown. For each metric, the name, the corresponding product, the timestamp, the value, and an indicator showing whether a metric has improved or worsened are given. In addition, the table is sorted by importance and the top entries are highlighted accordingly with different symbols. We calculate importance by normalizing the difference between the current and the previous measurement. The higher the difference, the more important a measurement is.

More information about a single metric can be investigated by a user with the information icon. When clicked the dialog presented in Figure 7.9 opens, which shows different insights about the metric, such as the definition, rationale, implication and a solution strategy to increase the metric performance. Finally past measurements of the metric are displayed in a line chart, so a user can see the development of the metric over time. Both the overview and the dialog were developed analogously for the other two pillars.

7. Tool-supported security maturity

Security Metrics

All automatic or manual metrics at a glance

Name	Application	Domain	Zeitpunkt	Value
! Architecture Component Attack Surface Analysis Rate i	📄 Test App	Design	1.12.	55% ↔
! Security Meetings Count i	👥 Team Metric	Effort	5.12.	12 ↓
💡 Security Requirements Threat Modeling Rate i	📄 Test App	Analysis	1.12.	47% ↑
💡 Security Requirements General Analysis Rate i	📄 Test App	Analysis	29.11.	43%
Omitted Security Requirements Count i	📄 Test App	Analysis	27.10.	12 ↑

Items per page: 5 1 - 5 of 11 |< < > |>

... Show all ✎ Manual Input

Figure 7.8.: An overview over all metric measurements making up the metric based pillar

Security Requirements Threat Modeling Rate

Product ↗ PMT Manual

Information about the Metric

Definition	Threat modeled security requirements / Amount of security requirements
Rationale	Percentage of security requirements covered by means of threat modeling and change impact analysis
Implication	Threat modeling gives an insight into what potential threats lie in the security requirements*
Solution Strategy	Reduce amount of security requirements (treating), increase security analysis time (solving)

Development of the Metric



Figure 7.9.: A dialog to support the user to understand a metric

7.4. Evaluation

In this section we describe the given feedback by the participants in three sections analogously to the used semi-structured questionnaire (see appendix A).

7.4.1. Security maturity dashboard

Firstly feedback was given by the experts on the visualization on the overall maturity score, the component diagram and the history chart as presented in Figure 7.7.

The experts considered the security maturity dashboard to be generally useful and a major evolution of the original Microsoft Excel based TSM. It was generally noted, that the dashboard gives you good information on where the team currently stands and what potential for improvements the team has at a glance, while still containing enough detailed information so a user or decision maker can track down information in specific domains or topics.

One expert noted, that the dashboard could be adapted to fit the information needs of a specific user group or security expert, as e.g. software engineers in the team require more product-focused information and concrete action items while managers often require more abstract security measures. He added that generally it can be assumed that these dashboards are used by expert users with some form of prior knowledge.

The overall score was seen as useful and not as oversimplifying, especially as the possibility of drilling down into the pillars and domains is given in the dashboard. One expert noted a school grade system with ordinal levels could be easier to interpret than a percentage with many small increments, as it is often used in the context of other security tools. He reasoned that it may be unclear what the actual consequences of a few percent increase in scale are. Another suggested solution was the introduction of a reference maturity score, which represents the enterprise's or business unit's target value, so the team can understand if their score matches the expectation or needs improvement.

Finally, feedback was given on the solution strategy to improve the maturity score. While the dashboard gives helpful information about the past development of the maturity score according to all experts, some noted that it would be helpful to receive more solution strategies and suggestions for improving the score in the future.

In addition to the broader aspects direct feedback was given by the experts on the look and feel of the first part of the dashboard and its components. Suggestions by the experts included:

- Highlighting domains that have a high or low score or which improved the most
- Improving the interactivity of the page by enabling clicking on a domain in the component chart and subsequently jumping to the corresponding assessments or metrics of that domain
- Translating the assessments and security metrics into German so the often complex assessment topics and metrics can be better understood
- Indicating maturity improvement or decreasing with corresponding colors

7. Tool-supported security maturity

The maturity dashboard was iteratively adapted to incorporate parts of the direct feedback. The result of the process can be seen in Figures 7.10 and 7.11, which show the initial dashboard and the current version.



Figure 7.10.: The first version of the maturity dashboard

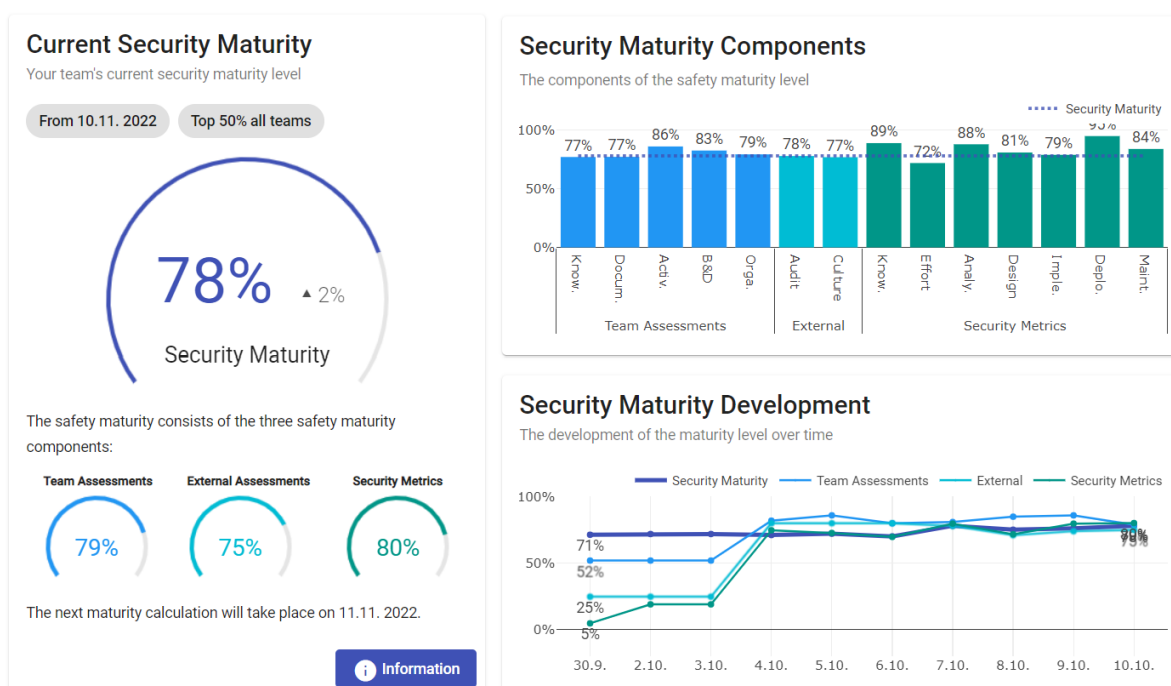


Figure 7.11.: The current version of the maturity dashboard

Additionally the experts gave feedback on the second part of the dashboard, which includes the overview over the metric measurements and carried out assessments. One expert noted that with more than 30 metrics and assessments an aggregation or automated sorting of the measurements would be necessary, so the team is not required to search for interesting

information themselves. Another expert noted, that it is difficult to deduce if a metric was measured with a good or insufficient value, as only the value is shown in the overview.

7.4.2. Input TSMC data sources

Feedback was also given by the experts on the collection and inputting of security data into the prototype.

Several experts noted that having many tools and having an easy and fast set up process for new security tools is vital for the acceptance of the tool. Two experts specifically mentioned they use SonarQube as a security tool in practice, which Prince already supports. One expert noted that it might be impossible to connect proprietary security software to the tool via an API, but since such a tool is essential to his security efforts, our proposed tool could not be adapted in its current version.

Another expert mentioned their experience with the constant change of specifications within the metrics of security tools. He stated that in the past the definition and calculation of many metrics changed and the measurements were not comparable any more.

For the assessments some experts mentioned the verbosity of the process. One participant described the implemented sliders as cumbersome because not all maturity levels are visible at the same time, and suggested a UI component such as radio buttons as a simpler solution. Another expert mentioned, that it is unclear when, how often and from whom assessments need to be carried out.

7.4.3. TSMC approach

The experts finally also gave feedback on the general applicability of the model and the prototype in their work environment. Generally all experts raised concerns regarding the assessment's verbosity, as one assessment exists of more than 30 topics with four different levels. To resolve this, one expert suggested the use of a small checkbox like survey, which could be completed in a higher frequency.

Additionally, one expert noted that from his experience and research the correlation between the answers in the self assessments and the actual quantitative measurements with metrics is often quite low. Additionally, he stated that quantitative metrics actually provide more reliable data. In his opinion, the introduction of security metrics for manual evaluation is very useful and should be carried out.

One expert pointed out the correlation between high-quality code and security, since in his experience security automatically increases when standard best practices are followed in software development. Additionally, the expert stated that this sort of discipline often does not introduce monetary costs or require much work time. He concluded that code quality measurements therefore should also be included in assessing the security capabilities of a team.

Some experts pointed out that the different topics, areas and pillars should be weighted differently depending on the environment. On the one hand, improvements or vulnerabilities that require a lot of work or pose a high risk of attack may have minimal impact on the

maturity score, which could lead to team frustration or respectively a false sense of security. An example of this scenario would be the discovery of the log4j vulnerability in December 2021⁴: in the TSMM only the “Open Vulnerabilities Count” would have increased by a single digit and subsequently the maturity level would have not noticeably worsened. On the other hand, two experts noted that different applications may have different needs for security, which should be taken into account when calculating the maturity of the team, i.e. by weighting the applications’ maturity scores accordingly. Also, since a good score in some areas might be currently undesired or unnecessary, the tool should offer the possibility to disable some components of the calculation.

One expert noted that in his experience development teams are often resistant about quantitative measurements, which could pose a problem for applying TSMM. To overcome this possible reluctance towards the approach, he suggests the development of a coaching program in which an external security coach analyses the TSMM performance and communicates it with developers in a non-threatening way.

Concerns were also raised by the experts that the approach presented introduces employee surveillance. While they see the development of secure applications and an active security monitoring tool for them as beneficial, they mentioned potential consequences of implementing the approach in their organization, such as monetary or cultural repercussions for teams with an insufficient maturity score. One expert therefore recommends thorough ethical and legal research before the tool is introduced into organizations.

One expert mentioned that special security activities are carried out in his organization that are currently not measured within the metric catalogue. As an example, he mentioned compliance with naming conventions in a public cluster with nodes from different customers and individual security certificates through on-site training.

Lastly, one expert mentioned the possibility of using the maturity score for security certification processes. His company relies on certifications to realise certain software engineering projects with high security needs, and the maturity score could help verify the security efforts undertaken.

⁴<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228>, accessed in November 2022

8. Discussion

In this chapter, we summarize our main findings and discuss their limitations. The results and limitations are structured analogously to the research artifacts.

8.1. Key finding and artifacts

From our research, we derived the following key findings, which we discuss in detail in the following sections:

1. Certain security metrics are meaning- and insightful
2. The automation level of security metrics with security tools varies
3. Security metrics can be used to calculate the team security maturity
4. A overall security maturity score can be calculated and may be helpful
5. Tool-supported security maturity involves a trade-off
6. The success of (tool-supported) security maturity depends on efficient information collection

Certain security metrics are meaning- and insightful

Measuring the team security level and the security levels of the developed artifacts is important as a feedback loop on the security efforts undertaken and as a tool for transparency and compliance. The security metrics we have collected in the security metrics catalogue have been systematically identified, evaluated against rigorous quality criteria, and allow us to measure a wide range of security areas during the SDLC. While the selection of appropriate security metrics can be made with respect to the environment security requirements, the catalogue provides a starting point for high quality metrics. Our evaluation showed that security metrics has real meaning and insight for team members such as developers, and for cross-team roles such as security specialists in a central security department. Self- or external assessments provide insightful information, but only at regular intervals, they are often voluminous, and they introduce a subjective component to the measurement. Measuring automated security metrics can solve these problems: they can be tracked continuously, require little manual effort, are objective and can be compared reliably over time.

The automation level of security metrics with security tools varies

Although security metrics often track small aspects of an application or team, it is still challenging to measure them with existing security tools. We found that eight metrics in our metrics catalogue cannot currently be measured automatically by any existing security tool, even though they provide insight into the development process. In practice, these metrics are therefore applicable only to a certain extent, as manual measurements of metrics are unreliable and require a lot of manual effort. On the other hand, some metrics are measurable with many security tools, which raises another issue. The same metric can be defined and calculated differently by different tools and subsequently the collected measurements are not comparable.

Security metrics can be used to calculate the team security maturity

Existing maturity models such as the TSMM often use self-assessments and external assessments as a source of information. Security metrics can complement this source of information and increase the quality of the security maturity calculation, because (semi-) automated security metrics provide relevant information, their measurements are quantifiable, and they can be measured with a high frequency. The introduction of security metrics also allows for a comparison between the perceived and actual security level of the developed artifacts. Nevertheless, we cannot completely replace manual assessments with security metrics because some relevant areas of software engineering, such as team culture, are currently not measurable with them.

A overall security maturity score can be calculated and may be helpful

In security maturity models, the focus lies often on assigning individual domains to their corresponding maturity levels in order to reveal security domains in need of improvement. However, the literature and our proposed calculation of a maturity score using the TSMM also emphasize that an overall score can be calculated across all security domains. Such a score greatly simplifies the underlying security data collected, and thus could theoretically mislead developers or security specialists. However, we showed that if properly calculated and presented, it can be used as a guidance tool by the team to accelerate security in agile development. Furthermore, it can be used as a reporting tool to visualize the security level in the developed applications to external security specialists.

Tool-supported security maturity involves a trade-off

Traditional maturity models are often still analog, meaning gathering information and assessing maturity is done through surveys or manual audits with spreadsheets. This includes the TSMM, which was the focus of this research. This type of information collection, while often thorough, is less reliable, requires a lot of time and effort, and is difficult to maintain. By implementing a dedicated security maturity measurement tool, many of these drawbacks can be mitigated. Firstly, security metrics can be collected automatically and

reliably without manual effort. Secondly, all maturity information, such as previous and current assessments, metrics, or maturity scores, are persistently stored in a structured format in a database. Finally, the collected data can be queried and visualized to gain new insights in the various security domains. These benefits were also confirmed by the experts in the conducted interviews. However, to achieve them, a trade-off must be made. Assessments are still used in our prototype to calculate the maturity score, which require a lot of manual effort because there are a large number of security topics. In addition, special security tools must be set up, which can measure the implemented security metrics.

8.2. Limitations

8.2.1. Validation of research methods [112, p.153]

To assess the threats to the validity of the proposed security metrics catalogue we consider the criteria of construct validity, internal validity, external validity and reliability in this Section [112, p.153]. The four concepts were proposed for validating case studies in software engineering research, but still relate to our research as we conducted a literature review and interviews with the developed prototype, where the typical limitations of case studies are also relevant.

Construct validity

Construct validity concerns a possible mismatch between the proposed artifacts and the research questions. Our goals were first to determine which security metrics exist in the literature, find out how they can be automatically captured by security tools, and second, how they can be used to assess the security maturity of a development team. To achieve the first goal, we conducted a literature review to create a structured security metrics catalogue. As new or applied security metrics are published in the scientific literature, this research methodology provides us with a holistic overview of existing security metrics and their level of automation. In addition, the literature review followed a clear structure as outlined in 3.1 and is therefore objective and reproducible. Nevertheless, with our chosen methodology, it is possible that some metrics are missing from our catalogue, as there are other sources of security metrics, such as security tools or video presentations.

To achieve our second research goal, we introduced the metrics we found into the TSMM. With the introduction and the calculation method presented in Section 6.2, we have demonstrated a way to assess the security maturity of a team with security metrics and thus fully answer our research question.

Internal validity

Internal validity must be investigated when causal relationships are studied in research, since it is possible that a factor under study is influenced not only by an observed factor but also

by another factor that is not monitored. Since we did not examine causal relationships, the issue of internal validity is not of concern in this research.

External validity

The topic of external validity investigates if and to what extent it is possible to generalize the findings from the conducted research. With intent, we proposed a general security metrics catalogue with many possible applications. One of these applications is the presented calculation of team security maturity, but it is very possible and encouraged, to investigate to use the catalogue as an audit tool, in small-scale agile efforts or as an implementation support tool. Additionally, it is possible to generalize the steps we conducted to assess the quality of security metrics, which includes applying minimal quality criteria and afterwards assigning metrics a score calculated from different quality aspects.

Reliability

Reliability discusses if the study results could be hypothetically reproduced by a different researcher later on or if they are dependent on the researcher. As the catalogue is the result of an extensive literature review, reliability is given until new metrics are proposed in the literature in the future. The introduction of the metrics into TSMM on the other hand shows only one way to answer the research question RQ2. It is feasible that at a later stage other possibilities will become apparent, e.g. the maturity could be calculated solely with security metrics without the need of any manual assessments.

8.2.2. Validation of proposed software prototype [75, p. 323]

In this section we assess the validity of the proposed prototype by using the recommendations of Lukyanenko et al., which focus explicitly on software artifacts [75, p.323].

Artifact instantiation space

The artifact instantiation space is the set of possible variations in the design of software features, which can pose a threat to validity because a theory can be instantiated with software in many ways and produce different results. In our case the prototype was developed in a large instantiation space, as became apparent during the development process and in the conducted interviews, where a lot of different user interface components and designs were suggested by the experts. It became apparent that our general system architecture design choices like the server-client architecture or the choice of framework is not in question, but the user interface design decisions in the frontend are ambiguous. In our case this large instantiation space therefore threatens the validity, as a different user interface design could have lead to much different interview results.

Artifact complexity

Auxiliary Features. This topic discusses requiring auxiliary data, a database or operating system to evaluate a software system, which can lead to a threat in instantiation validity. In our case the tool and therefore the evaluations required auxiliary data holding amongst others the maturity score of a team, which was chosen arbitrarily and could have an impact on the evaluation and acceptance decision. We mitigated this by explaining the experts, that the score is the result of an example calculation of a specific, fictional development team, which was generally well understood.

Emergent Properties. An example for emergent properties are presentation complexity or information overload while presenting information to the user. We believe our prototype does not invoke these properties, as became apparent in the evaluation by explaining it and asking specifically for unclear user interface components.

Artifact Cost

The effort to create a software artifact and consequently the difficulty of presenting multiple software versions is discussed by this topic. As our prototype was developed over the course of only a few months and only by one researcher, it was not possible to implement multiple software designs. Additionally, it was not possible to implement all features suggested by the experts and present the adapted version in a second round of interviews.

Artifact Medium and Distance

With software systems the theoretical construct and the implementation are often separated with a large distance e.g. in language and visualization aspects. In our case all assessments and security metrics are adopted from the TSMM, so from theory, and additionally all other aspects of the model such as the pillar structure was adopted and clearly visualized. Still the original table-like structure of the TSMM differs from the web application in many aspects, such as the visualization of the domains' maturity scores with an interactive diagram. Therefore, the approach with the TSMM and the web application should be to some extent evaluated separately, which we followed in the interviews. Still it is feasible, that the given evaluation was biased by the software design.

Technological Progress

This topic concerns the impact of a technological change on the proposed prototype. Although our proposed software architecture is currently state of the art, there are several issues that threaten its validity if the chosen technologies evolve. First, given the current shift to mobile, touch-based applications, the proposed prototype may need to be adapted and reevaluated [75, p.326]. Second, the chosen user interface structure may prove to be outdated or overly complex in the future and will need to be redesigned and reevaluated. In addition, new security tools may become established and some proposed security metrics may become drastically easier to measure. In this case, our approach to connecting to security tools via a REST API would need to be revised. We can therefore conclude that a future prototype with the same goal as ours will be designed, implemented, and behave differently.

9. Conclusion and Future Work

In this chapter we summarize the research. We answer the research questions described in Chapter 1 using the proposed artifacts and preview possible future work.

9.1. Summary

We created three artifacts, namely a security metrics catalogue, an adapted version of the TSMM and a maturity tool, that can be used independently or combined: For example, the security metrics catalogue can be used in an security metrics program without implementing the TSMM, and the TSMM can be used without the maturity tool. In what follows, we attempt to use these artifacts to provide a summary answer to each of our research questions.

RQ1: Which security metrics exist and how can they automatically be captured with the support of security tools?

To answer the question, we used a systematic literature review to create a catalogue of 31 qualifying metrics. The metrics collected have undergone a rigorous and structured qualification process and are therefore of high quality. In the catalogue, we described each metric in detail so that it is clear how it is calculated, what information it requires, and what solution strategies exist to increase its score. To analyze the automation potential of security metrics and thus answer the second part of the question, we identified common security tool categories and prominent examples of tools. Once identified, we connected the security metrics to the tools in a matrix that can serve as an exemplary guide to where and how each metric can be measured. The identified security metrics can be used to calculate security maturity, but are also of high value for general DevSecOps or any other efforts in secure software development.

RQ2: How can security metrics be used to assess the security maturity of an agile development team?

We integrated the identified security metrics into the TSMM to answer this research question. Together with the existing contents of the TSMM this yields a model that can be used to deterministically compute security maturity scores of different security domains. Additionally, we introduced the structured, multistep calculation of an overall maturity score across all domains of the TSMM. As a majority of the metrics can automatically be measured with security tools, the maturity score itself can also be calculated automatically, leading to faster and more reliable results.

RQ3: How can a team's security maturity be calculated, represented and visualized in a self-assessment tool?

We developed the functionality described in the research question into an existing self-assessment tool and therefore show through the description and screenshots of the tool a way to calculate, present and visualize the security maturity of a team. In addition, through the evaluation we conducted, we were able to demonstrate that the implemented functionality has real value and meets an information need. But the evaluation participants also provided suggestions for improvement, which we implemented in a second iteration of the prototype. The general concept and functionality of the tool could be used as a blueprint or kickstart for any other security maturity tool built on TSM or another maturity model (see Chapter 9.2).

9.2. Future work

In this section, we describe three opportunities for future research for which our work provide a foundation.

First, empirical research and validation is needed on our adapted version of the TSM. For the metrics-based pillar we proposed to complement the model, research should focus on validating the selection of security metrics and their respective maturity levels. In addition, the weighting of individual security metrics or domains in the metrics-based column could be investigated.

Second, the proposed security maturity tool should be researched and validated, which could focus on applicability in different industries, usefulness to different stakeholders and their information needs, or appropriateness at different security levels. In addition, we believe it would be useful to explore the use of metrics in other non-functional software domains such as quality, architecture, or user interface design. This stems from our literature review, where it became clear that there are many metrics outside of security. Options for conducting this research include a small workshop with synthetic security data, an industry case study with multiple development teams, or implementing the tool for a sample of teams and comparing their security capabilities to a control group over time.

It might also be useful to investigate the possibility of transferring our approach to other non-functional areas of software development, such as code quality, architecture, or user interface design. Firstly, a systematic literature review could be conducted to collect candidate metrics from the literature, after which the introduced qualification procedure could be applied. The qualified metrics then could be described with the same structured catalogue format. When the metrics are collected in this way, they could be introduced into a maturity model, similar to how we introduced security metrics into TSM, and eventually the proposed prototype could be used as a blueprint to collect the metrics and calculate maturity scores automatically.

A. Interview Questionnaire

Interviewee Information:

Position:

Job Description/ Experience:

Question Catalogue:

1. How is the security maturity score visualized in the security maturity dashboard?
 - **Maturity Score:** Is the meaning of the score and the difference between the different levels clearly displayed?
 - **Maturity Components:** Does it become clear how the maturity score is calculated? If not, what is missing or unclear?
 - What is your general, overall impression of the dashboard?
2. How is entering and querying data for TSMM performed?
 - **TSMM Assessments:** Does it become clear, how the different topics are answered in the system? Is the process straight-forward and engaging? If not, what is missing or unclear?
 - **Security Metrics:** Is it convenient and straight-forward how security metrics are collected? Is it easy to understand what security metrics are and how to interpret them? If not, what is missing or unclear?
 - What is your general, overall impression of entering TSMM data into the system?
3. What is your general feedback regarding this team security approach?
 - What advantages and benefits do you see for your team?
 - What disadvantages and drawbacks of the taken approach do you see?

B. Complete Catalogue of Security Metrics

B.1. Team security metrics

B.1.1. Knowledge

Name	Company Security Policy Review Rate	Level	derived
Entity	Company security policies	Type	internal
Attribute	Review	Range	[0,100]
Definition	Security policies reviewed / Amount of security policies	Expected Value	100
Rationale (theoretical)		Variability	[-10,10]
Percentage of relevant company security policies reviewed		Scale type	ratio
Implications (practical)		Related metrics	
Company security policy awareness is often low and increasing it can lead to more informed decision in early SDLC phases (shifts security left).		GSRRR	
Applicable in context		Validation	
Large enterprise with a central security unit		[22], [111], [62], [101]	
Solution Strategies			
Company security policy workshops, allow time for review			

B. Complete Catalogue of Security Metrics

Name	Government Security Regulation Review Rate	Level	derived
Entity	Government security regulations	Type	internal
Attribute	Review	Range	[0,100]
Definition	Government security regulations reviewed / Amount of government security regulations	Expected Value	100
Rationale (theoretical)		Variability	[-10,10]
Percentage of relevant government security regulation reviewed		Scale type	ratio
Implications (practical)		Related metrics	
Government security regulations need to be fulfilled to be on legal ground and avoid fines, but awareness is often low		CSPPR	
Applicable in context		Validation	
Large enterprise that handles sensitive personal data (e.g. necessary GDPR compliance)		[22], [111], [101]	
Solution Strategies			
Government security regulations training, allow time for review			

B.1.2. Velocity

Name	Security Awareness Effort Rate	Level	derived
Entity	Security awareness	Type	internal
Attribute	Effort	Range	[0,100]
Definition	Time spent on security awareness / Total development time	Expected Value	10
Rationale (theoretical)		Variability	[-5,5]
Security awareness is time spent on security training, review and similar activities		Scale type	ratio
Implications (practical)		Related metrics	
Time spend on awareness activities can be worth it in terms of efficiency and quality		SRER, SAPR, OSRC, MTTF	
Applicable in context		Validation	
Every software development context		[22], [42]	
Solution Strategies			
Offer training possibilities, allow time for security			

B. Complete Catalogue of Security Metrics

Name	Security Remediation Effort Rate	Level	derived
Entity	Security remediation	Type	internal
Attribute	Effort	Range	[0,100]
Definition	Time spent on security remediation / Total development time	Expected Value	10
Rationale (theoretical)		Variability	[-5,5]
Security remediation is time spent on implementing security requirements or fixing open vulnerabilities		Scale type	ratio
Implications (practical)		Related metrics	
Helps to measure if a team spends much time and if so if its efficiently		SMC, SAER, SAPR	
Applicable in context		Validation	
Every software development context		[125]	
Solution Strategies			
Offer training possibilities			

Name	Security Meetings Count	Level	derived
Entity	Milestone meetings	Type	external
Attribute	Security expert participation	Range	[0,∞]
Definition	Count meetings with security experts in an iteration	Expected Value	1
Rationale (theoretical)		Variability	$[-\infty, \infty]$
Amount of meetings with security experts participating per iteration		Scale type	absolute
Implications (practical)		Related metrics	
Security Experts should be involved early and often into the SDLC to provide guidance		SRER	
Applicable in context		Validation	
Large enterprise with a central security unit		[136]	
Solution Strategies			
Increase security expert participation in meetings			

B. Complete Catalogue of Security Metrics

Name	Security Audit Pass Rate	Level	derived
Entity	Security audits	Type	external
Attribute	Pass rate	Range	[0,100]
Definition	Passed security audits / Amount of security audits	Expected Value	100
Rationale (theoretical)		Variability	[-10,10]
Percentage of security audits successfully passed		Scale type	ratio
Implications (practical)		Related metrics	
Measure the security quality of the team's output		SAER, SRER	
Applicable in context		Validation	
Large enterprise with a central security unit		[136]	
Solution Strategies			
Reduce amount of security audits (treating), increase security efforts (solving)			

B.2. Product security metrics

B.2.1. Analysis

Name	Omitted Security Requirements Count	Level	base
Entity	Security requirements	Type	internal
Attribute	Omission	Range	[0,∞]
Definition	Count all omitted security requirements	Expected Value	0
Rationale (theoretical)		Variability	$[-\infty, \infty]$
Number of omitted security requirements from a standard that was fixed in advance (for instance, the Common Criteria for Information Technology Security Evaluation (ISO/IEC15408))		Scale type	absolute
Implications (practical)		Related metrics	
Security Requirements that are left are a threat to the security of the developed system		CSPPR	
Applicable in context		Validation	
Every software development context		[72], [42]	
Solution Strategies			
Decrease security requirements standard (treating), increase security awareness (solving)			

Name	Security Requirements General Analysis Rate	Level	derived
Entity	Security requirements	Type	internal
Attribute	Carried out analysis	Range	[0,100]
Definition	Analyzed security requirements / Amount of security requirements	Expected Value	100
Rationale (theoretical)		Variability	$[-5,5]$
Percentage of security requirements that have been subject to analysis prior to being included in the specification		Scale type	ratio
Implications (practical)		Related metrics	
Thorough analysis of the security requirements is beneficial to the security quality		SRTMR, SRSACR	
Applicable in context		Validation	
Every software development context		[78], [11], [7], [122], [42]	
Solution Strategies			
Reduce amount of security requirements (treating), increase security analysis time (solving)			

Name	Security Requirements Threat Modeling Rate	Level	derived
Entity	Security requirements	Type	internal
Attribute	Carried out threat modeling	Range	[0,100]
Definition	Threat modeled security requirements / Amount of security requirements	Expected Value	100
Rationale (theoretical)		Variability	[-5,5]
Percentage of security requirements covered by means of threat modeling and change impact analysis		Scale type	ratio
Implications (practical)		Related metrics	
Threat modeling gives an insight into what potential threats lie in the security requirements		SRGAR	
Applicable in context		Validation	
Every software development context with special requirements for system security		[78], [11], [106], [146], [111]	
Solution Strategies			
Reduce amount of security requirements (treating), increase security analysis time (solving)			

B.2.2. Design

Name	Architecture Component Attack Surface Analysis Rate	Level	derived
Entity	Architecture components	Type	internal
Attribute	Carried out attack surface analysis	Range	[0,100]
Definition	Attack surface analyzed architecture components / Amount of architecture components	Expected Value	100
Rationale (theoretical)		Variability	[-5,5]
Percentage of proposed architecture components subject to attack-surface analysis		Scale type	ratio
Implications (practical)		Related metrics	
Attack Surface Analysis gives insight into which areas have a higher exploitation risk than others		ACARAR	
Applicable in context		Validation	
Every software development context with special requirements for system security		[78], [101]	
Solution Strategies			
Decrease thoroughness of attack surface analysis (treating), increase security design time (solving)			

B. Complete Catalogue of Security Metrics

Name	Architecture Component Architectural Risk Analysis Rate	Level	derived
Entity	Architecture components	Type	internal
Attribute	Carried out risk analysis	Range	[0,100]
Definition	Architectural risk analyzed architecture components / Amount of architecture components	Expected Value	100
Rationale (theoretical)		Variability	[-5,5]
Percentage of proposed architecture components subject to architectural risk analysis		Scale type	ratio
Implications (practical)		Related metrics	
Architectural Risk provides a holistic view on flaws of the proposed architecture		ACASAR	
Applicable in context		Validation	
Every software development context with special requirements for system security		[78], [11], [109], [106], [6], [56], [101]	
Solution Strategies			
Decrease thoroughness of architectural risk analysis (treating), increase security design time (solving)			

Name	Security Requirements Satisfying Architecture Components Rate	Level	derived
Entity	Architecture components	Type	internal
Attribute	Security requirements satisfaction	Range	[0,100]
Definition	Architecture components with security requirements fulfillment / Amount of architecture components	Expected Value	100
Rationale (theoretical)		Variability	[-5,5]
Percentage of software components with demonstrated satisfaction of security requirements specifications		Scale type	ratio
Implications (practical)		Related metrics	
Architecture Components need to be mapped to the security requirements to have a high quality		SRGAR, UER	
Applicable in context		Validation	
Every software development context		[36], [77], [27], [101], [134]	
Solution Strategies			
Adhere stricter to security requirements in design phase			

B. Complete Catalogue of Security Metrics

Name	Coupling Corruption Propagation	Level	base
Entity	Architecture components	Type	internal
Attribute	Coupling corruption	Range	[0,∞]
Definition	MAX (amount of methods that could be affected by the originating unsecure method)	Expected Value	5
Rationale (theoretical)		Variability	[−∞,∞]
The total number of methods that could be affected by an originating unsecure method		Scale type	absolute
Implications (practical)		Related metrics	
When vulnerable a highly coupled system is more dangerous		CER	
Applicable in context		Validation	
Every software development context with special requirements for system security		[30], [77], [45], [120], [68], [5], [119], [19], [149], [41]	
Solution Strategies			
Follow a more island or decoupling system architecture approach			

Name	Critical Element Ratio	Level	derived
Entity	Architecture components	Type	internal
Attribute	Critical elements	Range	[0,100]
Definition	MAX (data elements which when corrupted have security impact / amount of data elements)	Expected Value	25
Rationale (theoretical)		Variability	[-1,1]
The rate of data elements which when corrupted or maliciously altered have security impact on the system to the total amount of data elements		Scale type	ratio
Implications (practical)		Related metrics	
Critical elements give an insight into the code structure		CCR	
Applicable in context		Validation	
Every software development context with special requirements for system security		[30], [77], [45], [120], [68], [19], [149], [41]	
Solution Strategies			
Introduce clear delimitation between critical and non-critical data elements in design			

B.2.3. Implementation

Name	Security Test Coverage Rate	Level	derived
Entity	System components	Type	internal
Attribute	Security tests	Range	[0,100]
Definition	Amount of system components covered by security tests / Amount of system components (e.g. LOC)	Expected Value	100
Rationale (theoretical)		Variability	[-1,1]
Security Test Coverage Rate is a measure of how much of a project's codebase is analyzed in the security tests performed		Scale type	ratio
Implications (practical)		Related metrics	
As many system components as possible should be tested to provide good insight into the security status of the system		STSR, STFR	
Applicable in context		Validation	
Every software development context		[141], [16], [79], [54], [142], [77], [76], [51], [25], [41], [53], [14]	
Solution Strategies			
Cover more parts of the system with security tests			

Name	Security Test Success Rate	Level	derived
Entity	Security tests	Type	internal
Attribute	Success	Range	[0,100]
Definition	Successful security tests / Total security tests	Expected Value	100
Rationale (theoretical)		Variability	[-1,1]
Percentage of how many security tests were successful out of all security tests		Scale type	ratio
Implications (practical)		Related metrics	
As many security tests as possible should pass to be confident about the system's security		STFR, STCR	
Applicable in context		Validation	
Every software development context		[11], [82], [141], [131], [91], [102], [6], [56], [142], [77], [76], [25], [41], [14]	
Solution Strategies			
Resolve underlying security problems			

B. Complete Catalogue of Security Metrics

Name	Security Test Failure Rate	Level	derived
Entity	Security tests	Type	internal
Attribute	Failure	Range	[0,100]
Definition	Failed security tests / Total security tests	Expected Value	0
Rationale (theoretical)		Variability	[-1,1]
Percentage of how many security tests failed out of all security tests		Scale type	ratio
Implications (practical)		Related metrics	
As few security tests as possible should fail to be confident about the system's security		STSR, STCR	
Applicable in context		Validation	
Every software development context		[11], [131], [6], [56], [47], [77], [76], [136], [25], [41], [14]	
Solution Strategies			
Resolve underlying security problems			

Name	Dependency Security Vulnerabilities Count	Level	base
Entity	Dependencies	Type	internal
Attribute	Security vulnerabilities	Range	[0,∞]
Definition	Count by dependencies introduced security vulnerabilities	Expected Value	0
Rationale (theoretical)		Variability	$[-\infty, \infty]$
The amount of vulnerabilities introduced by used dependencies		Scale type	absolute
Implications (practical)		Related metrics	
Vulnerable dependencies need to be removed or mitigated		DSVR	
Applicable in context		Validation	
Every software development context with introduced dependencies that are not enterprise internal		[109], [77]	
Solution Strategies			
Remove unnecessary dependencies, update outdated dependencies			

B. Complete Catalogue of Security Metrics

Name	Dependency Security Vulnerabilities Rate	Level	derived
Entity	Dependencies	Type	internal
Attribute	Security vulnerabilities	Range	[0,100]
Definition	Vulnerable dependencies / Amount of dependencies	Expected Value	0
Rationale (theoretical)		Variability	[-1,1]
The share of dependencies which introduce possible security vulnerabilities.		Scale type	ratio
Implications (practical)		Related metrics	
A single vulnerable dependencies can be catastrophic, even when the system has many		DSVC	
Applicable in context		Validation	
Every software development context with introduced dependencies that are not enterprise internal		[109], [77], [117]	
Solution Strategies			
Remove unnecessary dependencies, update outdated dependencies			

Name	Unsecured Endpoints Rate	Level	derived
Entity	Endpoints	Type	internal
Attribute	Insecurity	Range	[0,100]
Definition	Unsecure Endpoints / Amount of endpoints	Expected Value	0
Rationale (theoretical)		Variability	[-1,1]
The rate of endpoints without authorization control or other unsecure access possibilities		Scale type	ratio
Implications (practical)		Related metrics	
Endpoints are the attackers entry into a system		SRSACR	
Applicable in context		Validation	
Every software development context		[110], [149]	
Solution Strategies			
Follow stricter endpoint security practices			

B.2.4. Deployment

Name	Vulnerability Scanning Coverage Rate	Level	derived
Entity	System components	Type	internal
Attribute	Vulnerability scanning	Range	[0,100]
Definition	Amount of system components covered by vulnerability / Amount of system components (e.g. LOC)	Expected Value	100
Rationale (theoretical)		Variability	[-1,1]
The percentage of a software systems that have been checked for vulnerabilities		Scale type	ratio
Implications (practical)		Related metrics	
Only a holistically examination can give good insight		CSDR, OVC, MVC, VSR, CFR	
Applicable in context		Validation	
Every software development context		[148]	
Solution Strategies			
Introduce vulnerability scanning to more system components			

Name	Code Scanning Detection Rate	Level	derived
Entity	System components	Type	internal
Attribute	Vulnerable code scans	Range	[0,100]
Definition	Amount of problematic security scans / Total amount of security scans	Expected Value	0
Rationale (theoretical)		Variability	[-5,5]
Rate of security scans which identify a problem to the total amount of scans		Scale type	ratio
Implications (practical)		Related metrics	
Scans can give an objective view on the system and a good measure of the system's security		VSCR	
Applicable in context		Validation	
Every software development context with active periodic security scans		[82], [141]	
Solution Strategies			
Decrease the amount of scans (treating), increase the security efforts (solving)			

B. Complete Catalogue of Security Metrics

Name	Open Vulnerabilities Count	Level	base
Entity	System components	Type	internal
Attribute	Open vulnerabilities	Range	[0,∞]
Definition	Count all unmitigated vulnerabilities	Expected Value	0
Rationale (theoretical)		Variability	[−∞,∞]
The amount of open vulnerabilities in the developed artifact		Scale type	absolute
Implications (practical)		Related metrics	
A single open vulnerability can be catastrophic and mitigation should have a high priority		VSCR, OVR	
Applicable in context		Validation	
Every software development context		[22], [142], [47], [45], [84], [83], [41], [110], [109], [64], [79], [97], [102], [7], [120], [71], [56], [143], [4], [103], [77], [27], [136], [111], [51]	
Solution Strategies			
Weaken the vulnerability definition (treating), increase security efforts (solving)			

Name	Open Vulnerabilities Rate	Level	derived
Entity	System components	Type	internal
Attribute	Open vulnerabilities	Range	[0,100]
Definition	Amount of unmitigated vulnerabilities / System components (e.g. LOC)	Expected Value	0
Rationale (theoretical)		Variability	[-1,1]
The amount of open vulnerabilities of the system in relation to the size of the system		Scale type	ratio
Implications (practical)		Related metrics	
Vulnerabilities are more likely in complex system		VSCR, OVC	
Applicable in context		Validation	
Every software development context		[11], [141], [28], [97], [103], [36], [142], [47], [76], [72], [84], [122], [42], [41], [117]	
Solution Strategies			
Weaken the vulnerability definition (treating), increase security efforts (solving)			

B. Complete Catalogue of Security Metrics

Name	Mitigated Vulnerabilities Count	Level	base
Entity	System components	Type	internal
Attribute	Mitigated vulnerabilities	Range	[0,∞]
Definition	Count all already mitigated vulnerabilities	Expected Value	0
Rationale (theoretical)		Variability	$[-\infty, \infty]$
The amount of vulnerabilities which have been completely mitigated		Scale type	absolute
Implications (practical)		Related metrics	
Mitigation of vulnerabilities increases a team's experience and efficiency		MTTR	
Applicable in context		Validation	
Every software development context		[27], [83]	
Solution Strategies			
Weaken mitigation requirements (treating), increase security efforts (solving)			

Name	Vulnerability Slippage Rate	Level	derived
Entity	Live system components	Type	internal
Attribute	Vulnerabilities	Range	[0,100]
Definition	Amount of artifacts with vulnerabilities in production / Amount of Artifacts in production	Expected Value	0
Rationale (theoretical)		Variability	$[-1, 1]$
Rate of artifacts in production with included vulnerabilities		Scale type	ratio
Implications (practical)		Related metrics	
The presence of vulnerabilities in production indicate that the security efforts of the team are insufficient		VSCR	
Applicable in context		Validation	
Every software development context		[97], [6], [142], [89], [68], [83], [53], [53], [134], [2]	
Solution Strategies			
Increase security efforts			

B. Complete Catalogue of Security Metrics

Name	Change Failure Rate	Level	derived
Entity	Proposed change	Type	internal
Attribute	Failure	Range	[0,100]
Definition	Amount of proposed and failed changes / Amount of proposed changes	Expected Value	0
Rationale (theoretical)		Variability	[-5,5]
The amount of proposal artifacts which miss one or more quality gates		Scale type	ratio
Implications (practical)		Related metrics	
The presence of vulnerabilities in an artifact proposal indicate that the security efforts of the team are insufficient		VSCR	
Applicable in context		Validation	
Every software development context with active quality gates		[17], [13], [68], [51], [105]	
Solution Strategies			
Weaken change rejection criteria (treating), increase security efforts (solving)			

Name	Spillover Vulnerability Count	Level	base
Entity	New development iteration	Type	internal
Attribute	Vulnerabilities	Range	[0,∞]
Definition	Count all old and unmitigated vulnerabilities	Expected Value	0
Rationale (theoretical)		Variability	$[-\infty, \infty]$
Security vulnerabilities that do not get fixed during a given iteration		Scale type	absolute
Implications (practical)		Related metrics	
Increasing technical debt results in inefficiencies and reduced software quality			
Applicable in context		Validation	
Every software development context		[25], [2]	
Solution Strategies			
Increase security remediation efforts			

B.2.5. Maintenance

Name	Availability Rate	Level	derived
Entity	Production (run-time)	Type	internal
Attribute	Availability	Range	[0,100]
Definition	Time system was available / Total run time	Expected Value	100
Rationale (theoretical)		Variability	[-1,1]
The uptime/ downtime of the system		Scale type	ratio
Implications (practical)		Related metrics	
Availability has a direct impact on revenue, contractual service fulfillment or customer satisfaction		MTTR, MTTF	
Applicable in context		Validation	
Every software development context with some availability requirements		[27], [11], [8], [134], [14], [117], [2]	
Solution Strategies			
Keep system with detected insecurity online (treating), increase security efforts (solving)			

Name	Mean Time to Resolve	Level	derived
Entity	Vulnerability mitigation	Type	internal
Attribute	Velocity	Range	[0,∞]
Definition	AVG (Time it takes from vulnerability acknowledgement to mitigation)	Expected Value	24h
Rationale (theoretical)		Variability	$[-\infty, \infty]$
Mean time to resolve a discovered vulnerability during production		Scale type	ratio
Implications (practical)		Related metrics	
The velocity of vulnerability mitigation indicate if the team's security work is efficient		MVC, AR	
Applicable in context		Validation	
Every software development context		[8], [28], [59], [31], [91], [90], [64], [16], [54], [97], [102], [7], [88], [49], [6], [22], [143], [103], [142], [89], [76], [17], [111], [53], [101], [117], [105]	
Solution Strategies			
Increase security remediation efforts			

B. Complete Catalogue of Security Metrics

Name	Mean Time to Fail	Level	derived
Entity	Production (run-time)	Type	internal
Attribute	Security failure frequency	Range	[0,∞]
Definition	AVG (Time it takes for a vulnerability to be apparent/ the system is shut down because of an open security vulnerability)	Expected Value	∞
Rationale (theoretical)		Variability	[−∞,∞]
Mean time until a security failure happens in production		Scale type	ratio
Implications (practical)		Related metrics	
Vulnerabilities in production should happen as little as possible to avoid potential catastrophic consequences		AR, SAER	
Applicable in context		Validation	
Every software development context		[61], [111]	
Solution Strategies			
Weaken vulnerability definition (treating) , increase security efforts (solving)			

B.3. SMMM Calculation

			Quantitative					Qualitative				
			Quantifiable	Readiness	Repeatable	Cardinal	Score %	Measurability	Meaningfulness	Correctness	Score %	Total Score
Team metrics	Knowledge	CSPPR	1.0	1.0	1.0	1.0	100%	1.0	1.0	1.0	100%	100%
		GSRRR	1.0	1.0	1.0	1.0	100%	1.0	1.0	1.0	100%	100%
	Effort	SAER	0.5	0.5	1.0	1.0	75%	1.0	1.0	1.0	100%	88%
		SRER	0.5	0.5	1.0	1.0	75%	1.0	1.0	1.0	100%	88%
		SMC	1.0	0.5	1.0	1.0	88%	1.0	0.5	1.0	83%	85%
		SAPR	1.0	0.5	1.0	1.0	88%	1.0	1.0	1.0	100%	94%
	Analysis	OSRC	1.0	1.0	1.0	1.0	100%	1.0	0.5	1.0	83%	92%
		SRGAR	0.5	1.0	1.0	1.0	88%	0.5	1.0	1.0	83%	85%
		SRTMR	1.0	1.0	1.0	1.0	100%	1.0	1.0	1.0	100%	100%
	Design	ACASAR	1.0	0.5	1.0	1.0	88%	1.0	1.0	1.0	100%	94%
ACARAR		1.0	0.5	1.0	1.0	88%	1.0	1.0	1.0	100%	94%	
SRSACR		1.0	1.0	1.0	1.0	100%	0.5	1	1.0	83%	92%	
CCR		0.5	1.0	1.0	1.0	88%	0.5	1.0	1.0	83%	85%	
CER		0.5	1.0	1.0	1.0	88%	0.5	1.0	1.0	83%	85%	
Product metrics	Implementation	STCR	1.0	1.0	1.0	1.0	100%	1.0	0.5	1.0	83%	92%
		STSR	1.0	1.0	1.0	1.0	100%	1.0	0.5	1.0	83%	92%
		STFR	1.0	1.0	1.0	1.0	100%	1.0	0.5	1.0	83%	92%
		DSVC	1.0	0.5	1.0	1.0	88%	1.0	1.0	1.0	100%	94%
		DSVR	0.5	1.0	1.0	1.0	88%	1.0	1.0	1.0	100%	94%
		UER	1.0	1.0	1.0	1.0	100%	1.0	1.0	1.0	100%	100%
	Deployment	VSCR	1.0	1.0	1.0	1.0	100%	1.0	1.0	1.0	100%	100%
		CSDR	1.0	1.0	1.0	1.0	100%	1.0	1.0	1.0	100%	100%
		OVC	1.0	1.0	1.0	1.0	100%	1.0	0.5	1.0	83%	92%
		OVR	0.5	1.0	1.0	1.0	88%	1.0	1.0	1.0	100%	94%
MVC		1.0	1.0	1.0	1.0	100%	1.0	0.5	1.0	83%	92%	
VSR		0.5	0.5	1.0	1.0	75%	1.0	1.0	1.0	100%	88%	
CFR		1.0	1.0	1.0	1.0	100%	1.0	1.0	1.0	100%	100%	
SDC		1.0	1.0	1.0	1.0	100%	1.0	0.5	1.0	83%	92%	
Maintenance	AR	1.0	1.0	1.0	1.0	100%	1.0	1.0	1.0	100%	100%	
	MTTR	1.0	0.5	1.0	1.0	88%	1.0	1.0	1.0	100%	94%	
	MTTF	1.0	0.5	1.0	1.0	88%	0.5	1.0	1.0	83%	85%	

Glossary

- DAST** Dynamic application security testing. 10, 21, 24, 25
- DevOps** Development operations. 5, 6
- DevSecOps** Development security operations. 6, 8, 10, 54
- DSPR** Design science research process. 4
- IAST** Interactive application security testing. 10, 24, 25
- PMT** Project management tool. 11, 21, 22, 23, 24, 25, 26
- Prince** Principle self-assessment tool for agile development teams. 33, 34, 35, 39, 40, 41, 42, 47
- RQ1** Research question 1. 3, 20, 54
- RQ2** Research question 2. 3, 52, 54
- RQ3** Research question 3. 3, 33, 55
- SAST** Static application security testing. 10, 21, 22, 24, 25, 34, 41
- SCA** Software composition analysis. 10, 11, 21, 24, 25, 34
- SDLC** Software development life cycle. 5, 6, 8, 10, 14, 17, 18, 20, 21, 22, 23, 24, 25, 33, 49
- Security metrics** Secure software engineering metrics. 1, 3, 5, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 33, 34, 35, 36, 39, 41, 42, 43, 45, 47, 49, 50, 51, 52, 53, 54, 55
- SIEM** Security information and event management. 10, 11, 21, 25, 26
- SMMM** Security metrics maturity model. 9, 14, 19, 21, 23
- TSMM** Team security maturity model. 2, 3, 7, 8, 18, 21, 28, 29, 30, 31, 33, 34, 36, 39, 40, 42, 45, 47, 48, 50, 51, 52, 53, 54, 55
- USE** User-centric software engineering. 33, 34
- VMT** Vulnerability management tool. 10, 11, 22, 24, 25

List of Figures

- 1.1. Cyber incidents in the last years 2
- 2.1. The five phases of the SDLC 6
- 5.1. Security metrics catalogue distributions 21
- 6.1. A high-level TSMM Overview 29
- 7.1. Security Metrics in the backend 35
- 7.2. Maturity calculation in the backend 36
- 7.3. Prototype architecture 39
- 7.4. Assessments in the prototype 40
- 7.5. Setup of a metric source 41
- 7.6. Manual metric input in the prototype 42
- 7.7. Prototype dashboard 43
- 7.8. Metrics overview 44
- 7.9. Metric information dialog 44
- 7.10. The first version of the maturity dashboard 46
- 7.11. The current version of the maturity dashboard 46

List of Tables

2.1. An exemplary TSMM topic	8
2.2. An exemplary SMMM calculation	9
3.1. Literature review search strings	13
3.2. Publications in the literature	13
3.3. Unqualified candidate metrics	14
3.4. Interviewee information	16
5.1. Knowledge metrics	22
5.2. Effort metrics	22
5.3. Analysis metrics	23
5.4. Design metrics	24
5.5. Implementation metrics	24
5.6. Deployment metrics	25
5.7. Maintenance metrics	26
5.8. Measurement of security metrics with exemplary tools	27
6.1. Exemplary description of a security metric	29
6.2. Maturity values for an exemplary maturity score calculation	31
7.1. Supported tools in the prototype	34

Bibliography

- [1] R. Achatz and F. Paulisch. "Industrial strength software and quality: software and engineering at Siemens." In: *Third International Conference on Quality Software, 2003. Proceedings*. IEEE, 2003. DOI: 10.1109/qsic.2003.1319117.
- [2] A. Adomavicius. *Go beyond velocity with advanced product metrics*. DevBridge. <https://www.devbridge.com/white-papers/advanced-product-metrics/establish-framework/>. July 2020.
- [3] Y. Ahmed, S. Naqvi, and M. Josephs. "Aggregation of security metrics for decision making: a reference architecture." In: *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. 2018, pp. 1–7.
- [4] A. Al-Far, A. Qusef, and S. Almajali. "Measuring Impact Score on Confidentiality, Integrity, and Availability Using Code Metrics." In: *2018 International Arab Conference on Information Technology (ACIT)*. 2018, pp. 1–9. DOI: 10.1109/ACIT.2018.8672678.
- [5] B. Alshammari, C. Fidge, and D. Corney. "Security Metrics for Java Bytecode Programs." In: *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*. 2013, pp. 394–399.
- [6] S. A. Ansar, Alka, and R. A. Khan. "A Phase-wise Review of Software Security Metrics." In: *Networking Communication and Data Knowledge Engineering*. Springer Singapore, Nov. 2017, pp. 15–25. DOI: 10.1007/978-981-10-4600-1_2.
- [7] A. Arabsorkhi and F. Ghaffari. "Security Metrics: Principles and Security Assessment Methods." In: *2018 9th International Symposium on Telecommunications (IST)*. IEEE, Dec. 2018. DOI: 10.1109/istel.2018.8661030.
- [8] Atlassian. *How to choose incident management KPIs and metrics*. <https://www.atlassian.com/incident-management/kpis>.
- [9] Atlassian. *REST APIs*. <https://developer.atlassian.com/server/jira/platform/rest-apis/>.
- [10] R. Barabanov, S. Kowalski, and L. Yngström. *Information Security Metrics: State of the Art*. Jan. 2011.
- [11] N. Bartol and B. A. Hamilton. *Practical Measurement Framework for Software Assurance and Information Security*. Tech. rep. Practical Software and Systems Measurement, 2008.
- [12] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. *Manifesto for Agile Software Development*. <https://agilemanifesto.org/principles.html>. 2001.

- [13] Y. Beres, M. C. Mont, J. Griffin, and S. Shiu. "Using security metrics coupled with predictive modeling and simulation to assess security processes." In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, Oct. 2009. DOI: 10.1109/esem.2009.5314213.
- [14] O. Beyond. *How We Measure Software Quality*. One Beyond. <https://www.one-beyond.com/how-we-measure-software-quality/>. 2018.
- [15] S. Bhatt, P. K. Manadhata, and L. Zomlot. "The Operational Role of Security Information and Event Management Systems." In: *IEEE Security & Privacy* 12.5 (2014), pp. 35–41. DOI: 10.1109/msp.2014.103.
- [16] S. Bhattacharya, M. P. Singh, and L. Williams. "Software Security Readiness and Deployment." In: *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, Oct. 2021. DOI: 10.1109/issrew53611.2021.00088.
- [17] S. J. Bigelow. *10 DevSecOps metrics that actually measure success*. <https://www.techtarget.com/searchitoperations/tip/10-DevSecOps-metrics-that-actually-measure-success>. Dec. 2021.
- [18] R. Bongard, K. Dussa-Zieger, R. Reissing, and A. Schulz. *Basiswissen Automotive Softwaretest*. Dpunkt.Verlag GmbH, Sept. 2020. ISBN: 386490580X.
- [19] L. Borodaev, A. Telea, R. Groenboom, and R. Smedinga. "Software Metrics for Policy-Driven Software Development Life Cycle Automation." In: *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2018. DOI: 10.1109/icstw.2018.00047.
- [20] E. Bouwers, A. v. Deursen, and J. Visser. "Towards a Catalog Format for Software Metrics." In: *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics*. WETSoM 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 4447. ISBN: 9781450328548. DOI: 10.1145/2593868.2593876.
- [21] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. "Lessons from applying the systematic literature review process within the software engineering domain." In: *Journal of Systems and Software* 80.4 (Apr. 2007), pp. 571–583. DOI: 10.1016/j.jss.2006.07.009.
- [22] W. K. Brothby and G. Hinson. *PRAGMATIC Security Metrics*. Taylor & Francis Ltd., Apr. 2016. 512 pp. ISBN: 1439881537.
- [23] A. Brucker and U. Sodan. "Deploying static application security testing on a large scale." In: *Sicherheit 2014 Sicherheit, Schutz und Zuverlässigkeit*. Ed. by S. Katzenbeisser, V. Lotz, and E. Weippl. Bonn: Gesellschaft für Informatik e.V., 2014, pp. 91–101.
- [24] T. J. Carter and D. Dunning. "Faulty Self-Assessment: Why Evaluating Ones Own Competence Is an Intrinsically Difficult Task." In: *Social and Personality Psychology Compass* 2.1 (Nov. 2007), pp. 346–360. DOI: 10.1111/j.1751-9004.2007.00031.x.

- [25] K. Chakravarty and J. Singh. "A Study of Quality Metrics in Agile Software Development." In: *Machine Learning and Information Processing*. Springer Singapore, 2021, pp. 255–266. DOI: 10.1007/978-981-33-4859-2_26.
- [26] Y. Cheng, J. Deng, J. Li, S. A. DeLoach, A. Singhal, and X. Ou. "Metrics of Security." In: *Advances in Information Security*. Springer International Publishing, 2014, pp. 263–295. DOI: 10.1007/978-3-319-11391-3_13.
- [27] E. Chew, M. Swanson, K. Stine, N. Bartol, A. Brown, and W. Robinson. "Performance measurement guide for information security." In: *SP 800-55 Rev. 1* (2008).
- [28] E. Chickowski. *Seven Winning DevSecOps Metrics Security Should Track*. <https://businesinsights.bitdefender.com/seven-winning-devsecops-metrics-security-should-track>. 2018.
- [29] J. Choliz, J. Vilas, and J. Moreira. "Independent Security Testing on Agile Software Development: A Case Study in a Software Company." In: *2015 10th International Conference on Availability, Reliability and Security*. IEEE, 2015. DOI: 10.1109/ares.2015.79.
- [30] I. Chowdhury, B. Chan, and M. Zulkernine. "Security Metrics for Source Code Structures." In: *Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems*. SESS '08. Leipzig, Germany: Association for Computing Machinery, 2008, pp. 57–64. ISBN: 9781605580425. DOI: 10.1145/1370905.1370913.
- [31] A. Crouch. *DevSecOps: Incorporate Security into DevOps to Reduce Software Risk*. Tech. rep. Agile Connection, 2017.
- [32] CSIS. *Significant Cyber Incidents Since 2006*. Tech. rep. Center for Strategy & International Studies, 2022.
- [33] N. Davis, W. Humphrey, S. Redwine, G. Zibulski, and G. McGraw. "Processes for producing secure software." In: *IEEE Security & Privacy Magazine* 2.3 (2004), pp. 18–25. DOI: 10.1109/msp.2004.21.
- [34] M. DeBellis and C. Haapala. "User-centric Software Engineering." In: *IEEE Expert* 10.1 (1995), pp. 34–41. DOI: 10.1109/64.391959.
- [35] *Defect Dojo Resource Center*. <https://www.defectdojo.com/resources>.
- [36] C. Dekkers, D. Zubrow, and J. McCurley. *Measures and Measurement for Secure Software Development*. Tech. rep. Software Engineering Institute, Carnegie Mellon University, 2007.
- [37] *Dependency Track Introduction*. <https://docs.dependencytrack.org/>.
- [38] digital.ai. *15th Annual State Of Agile Report*. <https://digital.ai/resource-center/analyst-reports/state-of-agile-report>. 2021.
- [39] K. Dikert, M. Paasivaara, and C. Lassenius. "Challenges and success factors for large-scale agile transformations: A systematic literature review." In: *Journal of Systems and Software* 119 (Sept. 2016), pp. 87–108. DOI: 10.1016/j.jss.2016.06.013.

- [40] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe. "A decade of agile methodologies: Towards explaining agile software development." In: *Journal of Systems and Software* 85.6 (2012), pp. 1213–1221. doi: 10.1016/j.jss.2012.02.033.
- [41] G. Dlamini, S. Ergasheva, Z. Kholmatova, A. Kruglov, A. Sadovykh, G. Succi, A. Timchenko, X. Vasquez, and E. Zouev. "Metrics for Software Process Quality Assessment in the Late Phases of SDLC." In: *Lecture Notes in Networks and Systems*. Springer International Publishing, 2022, pp. 639–655. doi: 10.1007/978-3-031-10461-9_44.
- [42] S. Ergasheva and A. Kruglov. "Software Development Life Cycle early phases and quality metrics: A Systematic Literature Review." In: *Journal of Physics: Conference Series* 1694.1 (2020), p. 012007. doi: 10.1088/1742-6596/1694/1/012007.
- [43] T. Fertig, A. Schütz, and K. Weber. "Current Issues Of Metrics For Information Security Awareness." In: *Proceedings of the Twenty-Eighth European Conference on Information Systems*. June 2020.
- [44] C. Fry and S. Greene. "Large Scale Agile Transformation in an On-Demand World." In: *AGILE 2007 (AGILE 2007)*. IEEE, 2007. doi: 10.1109/agile.2007.38.
- [45] S. Ganesh, T. Ohlsson, and F. Palma. "Predicting Security Vulnerabilities using Source Code Metrics." In: *2021 Swedish Workshop on Data Science (SweDS)*. IEEE, Dec. 2021. doi: 10.1109/sweds53855.2021.9638301.
- [46] V. Garousi, M. Felderer, and M. V. Mäntylä. "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering." In: *Information and Software Technology* 106 (Feb. 2019), pp. 101–121. doi: 10.1016/j.infsof.2018.09.006.
- [47] M. Gomes, R. Pereira, M. Silva, J. B. de Vasconcelos, and Á. Rocha. "KPIs for Evaluation of DevOps Teams." In: *World Conference on Information Systems and Technologies*. Springer, 2022, pp. 142–156.
- [48] G. Granadillo, S. Zarzosa, and R. Diaz. "Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures." In: *Sensors* 21.14 (2021), p. 4759. doi: 10.3390/s21144759.
- [49] GSA. *DevSecOps Guide - Standard DevSecOps Platform Framework*. https://tech.gsa.gov/guides/dev_sec_ops_guide/.
- [50] D. Gupta and I. Khan. "Security Metrics Expectation and Reality." In: *International Journal of Advanced Research in Computer Science and Software Engineering* 5 (Mar. 2015), pp. 625–629.
- [51] P. Haindl and R. Plösch. "Value-oriented quality metrics in software development: Practical relevance from a software engineering perspective." In: *IET Software* 16.2 (2021), pp. 167–184. doi: 10.1049/sfw2.12051.

- [52] A. van der Heijden, C. Broasca, and A. Serebrenik. "An empirical perspective on security challenges in large-scale agile software development." In: *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, Oct. 2018. DOI: 10.1145/3239235.3267426.
- [53] S. S. Hossain, P. Ahmed, and Y. Arafat. "Software Process Metrics in Agile Software Development: A Systematic Mapping Study." In: *Computational Science and Its Applications – ICCSA 2021*. Springer International Publishing, 2021, pp. 15–26. DOI: 10.1007/978-3-030-87013-3_2.
- [54] S. Islam and P. Falcarin. "Measuring security requirements for software security." In: *2011 IEEE 10th International Conference on Cybernetic Intelligent Systems (CIS)*. IEEE, Sept. 2011. DOI: 10.1109/cis.2011.6169137.
- [55] I. Jacobson. *The unified software development process*. Pearson Education India, 1999.
- [56] S. Jain and M. Ingle. "Security Metrics and Software Development Progression." In: *Int. Journal of Engineering Research and Applications* (2014).
- [57] W. Jansen. "Directions in Security Metrics Research." In: *NISTIR 7564* (2009).
- [58] A. Jaquith. *Security Metrics. Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional, 2007, p. 336. ISBN: 9780321349989.
- [59] A. Jerbi. *KPIs for managing and optimizing devsecops success*. <https://www.infoworld.com/article/3237046/kpis-for-managing-and-optimizing-devsecops-success.html>. 2017.
- [60] P. Johnson, H. Kou, M. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita. "Improving software development management through software project telemetry." In: *IEEE Software* 22.4 (2005), pp. 76–85. DOI: 10.1109/MS.2005.95.
- [61] E. Jonsson and L. Pirzadeh. "A Framework for Security Metrics Based on Operational System Attributes." In: *2011 Third International Workshop on Security Measurements and Metrics*. IEEE, Sept. 2011. DOI: 10.1109/metrisec.2011.19.
- [62] E. Kahraman. *Evaluating IT security performance with quantifiable metrics*. July 2008.
- [63] H. Kallio, A.-M. Pietilä, M. Johnson, and M. Kangasniemi. "Systematic methodological review: developing a framework for a qualitative semi-structured interview guide." In: *Journal of Advanced Nursing* 72.12 (2016), pp. 2954–2965. DOI: 10.1111/jan.13031.
- [64] S. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Professional, 2002.
- [65] T. Karvonen, P. Rodriguez, P. Kuvaja, K. Mikkonen, and M. Oivo. "Adapting the Lean Enterprise Self-Assessment Tool for the Software Development Domain." In: *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. 2012, pp. 266–273. DOI: 10.1109/SEAA.2012.51.
- [66] K. Kaur, A. Jajoo, and Manisha. "Applying Agile Methodologies in Industry Projects: Benefits and Challenges." In: *2015 International Conference on Computing Communication Control and Automation*. IEEE, 2015. DOI: 10.1109/iccubea.2015.166.

- [67] D. Kengo Oka. "Software Composition Analysis in the Automotive Industry." In: *Building Secure Cars: Assuring the Automotive Software Development Lifecycle*. 2021, pp. 91–110. DOI: 10.1002/9781119710783.ch6.
- [68] R. A. Khan, S. U. Khan, and M. Ilyas. "Exploring Security Procedures in Secure Software Engineering: A Systematic Mapping Study." In: *The International Conference on Evaluation and Assessment in Software Engineering 2022*. ACM, June 2022. DOI: 10.1145/3530019.3531336.
- [69] S. Kim, H. Lee, Y. Kwon, M. Yu, and H. Jo. "Our Journey to Becoming Agile: Experiences with Agile Transformation in Samsung Electronics." In: *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2016. DOI: 10.1109/apsec.2016.064.
- [70] G. Klimko. "Knowledge Management and Maturity Models: Building Common Understanding." In: *Second European Conference on Knowledge Management*. 2001.
- [71] S. R. T. Kumar, A. Sumithra, and K. Alagarsamy. "The Applicability of Existing Metrics for Software Security." In: *International Journal of Computer Applications* 8.2 (Oct. 2010), pp. 29–33. DOI: 10.5120/1184-1638.
- [72] M. Kundi and R. Chitchyan. "Position on metrics for security in requirements engineering." In: *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*. IEEE, Aug. 2014. DOI: 10.1109/ret.2014.6908676.
- [73] M. B. Line, O. Nordland, L. Røstad, and I. A. Tøndel. "Safety vs security?" In: *PSAM Conference, New Orleans, USA*. sn. 2006.
- [74] A. S. S. Ltd. *Shift Left DevOps*. <https://www.aquasec.com/cloud-native-academy/devsecops/shift-left-devops/>. 2022.
- [75] R. Lukyanenko, J. Evermann, and J. Parsons. "Instantiation Validity in IS Design Research." In: *Advancing the Impact of Design Science: Moving from Theory to Practice*. Springer International Publishing, 2014, pp. 321–328. DOI: 10.1007/978-3-319-06701-8_22.
- [76] M. Maddox and S. Walker. "Agile Software Quality Metrics." In: *2021 IEEE MetroCon*. IEEE, Nov. 2021. DOI: 10.1109/metrocon54219.2021.9666049.
- [77] W. Mallouli, A. Cavalli, A. Bagnato, and E. M. de Oca. "Metrics-driven DevSecOps." In: *Proceedings of the 15th International Conference on Software Technologies*. SCITEPRESS - Science and Technology Publications, 2020. DOI: 10.5220/0009889602280233.
- [78] N. R. Mead and C. C. Woody. *Cyber security engineering: a practical approach for systems and software assurance*. Addison-Wesley, 2017.
- [79] N. Medeiros, N. Ivaki, P. Costa, and M. Vieira. "Software Metrics as Indicators of Security Vulnerabilities." In: *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Oct. 2017. DOI: 10.1109/issre.2017.11.
- [80] P. H. Meland and J. Jensen. "Secure Software Design in Practice." In: *2008 Third International Conference on Availability, Reliability and Security*. IEEE, 2008. DOI: 10.1109/ares.2008.48.

- [81] A. Meneely, B. Smith, and L. Williams. "Validating software metrics." In: *ACM Transactions on Software Engineering and Methodology* 21.4 (Nov. 2012), pp. 1–28. DOI: 10.1145/2377656.2377661.
- [82] MicroFocus. *Measuring DevOps Success*. Tech. rep. MicroFocus, 2018.
- [83] T. Mladenova. "Software Quality Metrics – Research, Analysis and Recommendation." In: *2020 International Conference Automatics and Informatics (ICAI)*. IEEE, Oct. 2020. DOI: 10.1109/icai50593.2020.9311361.
- [84] P. Morrison, D. Moye, R. Pandita, and L. Williams. "Mapping the field of software life cycle security metrics." In: *Information and Software Technology* 102 (Oct. 2018), pp. 146–159. DOI: 10.1016/j.infsof.2018.05.011.
- [85] S. M. Muthukrishnan and S. Palaniappan. "Security metrics maturity model for operational security." In: *2016 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. IEEE, May 2016. DOI: 10.1109/iscaie.2016.7575045.
- [86] A. M. Myers. "The clinical Swiss army knife. Empirical evidence on the validity of IADL functional status measures." In: *Med Care* 30.5 Suppl (May 1992), pp. 96–111.
- [87] A. M. Myers, P. J. Holliday, K. A. Harvey, and K. S. Hutchinson. "Functional Performance Measures: Are They Superior to Self-Assessments?" In: *Journal of Gerontology* 48.5 (Sept. 1993), pp. M196–M206. DOI: 10.1093/geronj/48.5.m196.
- [88] B. Nichols. *The Current State of DevSecOps Metrics*. Carnegie Mellon University's Software Engineering Institute Blog. <http://insights.sei.cmu.edu/blog/the-current-state-of-devsecops-metrics/>. Mar. 2021.
- [89] W. R. Nichols, H. Yasar, L. Antunes, C. L. Miller, and R. McCarthy. *Automated Data for DevSecOps Programs*. Tech. rep. Acquisition Research Program, 2022.
- [90] D. Nicolette. *Software Development Metrics*. MANNING PUBN, Aug. 2015. 192 pp. ISBN: 1617291358.
- [91] J. B. Norman Fenton. *Software Metrics - a rigorous and practical approach*. Taylor & Francis Ltd., Oct. 2014. 617 pp. ISBN: 1439838232.
- [92] E. Novikova and I. Kotenko. "Analytical Visualization Techniques for Security Information and Event Management." In: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 2013. DOI: 10.1109/pdp.2013.84.
- [93] D. Oezkan and A. Mishra. "Agile Project Management Tools: A Brief Comparative View." In: *Cybernetics and Information Technologies* 19.4 (2019), pp. 17–25. DOI: 10.2478/cait-2019-0033.
- [94] M. M. Olama and J. Nutaro. "Secure it now or secure it later: the benefits of addressing cyber-security from the outset." In: *SPIE Proceedings*. Ed. by I. V. Ternovskiy and P. Chin. SPIE, 2013. DOI: 10.1117/12.2015465.
- [95] *OSSIM Security Resources*. https://cybersecurity.att.com/resource-center#product_ossim.

- [96] OWASP ZAP Documentation. <https://www.zaproxy.org/docs/>.
- [97] K. V. J. Padmini, H. M. N. D. Bandara, and I. Perera. "Use of software metrics in agile software development process." In: *2015 Moratuwa Engineering Research Conference (MERCon)*. IEEE, Apr. 2015. doi: 10.1109/mercon.2015.7112365.
- [98] Y. Pan. "Interactive Application Security Testing." In: *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*. 2019, pp. 558–561. doi: 10.1109/ICSGEA.2019.00131.
- [99] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. "A Design Science Research Methodology for Information Systems Research." In: *Journal of Management Information Systems* 24.3 (Dec. 2007), pp. 45–77. doi: 10.2753/mis0742-1222240302.
- [100] R. Pemberton, E. Li, W. Or, and H. Pierson. *Taking Control: Autonomy in Language Learning*. Hong Kong University Press, 1996. ISBN: 9789622094079.
- [101] S. M. Poremba. *CIS to release consensus IT security metrics*. <https://www.scmagazine.com/news/breach/cis-to-release-consensus-it-security-metrics>. 2008.
- [102] S. Pradhan, V. Nanniyur, P. Melanahalli, M. Palla, and S. Chulani. "Quality Metrics for Hybrid Software Development Organizations – A Case Study." In: *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, July 2019. doi: 10.1109/qrs-c.2019.00097.
- [103] L. Prates, J. Faustino, M. Silva, and R. Pereira. "DevSecOps Metrics." In: *Information Systems: Research, Development, Applications, Education*. Springer International Publishing, 2019, pp. 77–90. doi: 10.1007/978-3-030-29608-7_7.
- [104] S. Radack. "Security Metrics: Measurements to Support the Continued Development of Information Security Technology." In: *ITL Bulletin* (2010).
- [105] N. Radzuan. *DevSecOps Transformation Bucket List*. medium.com. <https://medium.com/devops4me/devsecops-transformation-bucket-list-67d0873e283d>. Apr. 2022.
- [106] M. Rao. *Building your DevSecOps pipeline: 5 essential activities*. <https://www.synopsys.com/blogs/software-security/devsecops-pipeline-checklist/>. 2017.
- [107] RedHat. *What is DevSecOps?* <https://www.redhat.com/en/topics/devops/what-is-devsecops>. Apr. 2018.
- [108] K. Rindell, J. Ruohonen, J. Holvitie, S. Hyrynsalmi, and V. Leppänen. "Security in agile software development: A practitioner survey." In: *Information and Software Technology* 131 (Mar. 2021), p. 106488. doi: 10.1016/j.infsof.2020.106488.
- [109] C. Romeo. *The 3 most crucial security behaviors in DevSecOps*. <https://techbeacon.com/app-dev-testing/3-most-crucial-security-behaviors-devsecops>. 2019.
- [110] M. Roy. *5 IT Security Metrics That Matter Across Frameworks*. <https://www.axonius.com/blog/5-it-security-metrics-that-matter-across-frameworks>. Aug. 2021.
- [111] M. Rudolph and R. Schwarz. "A Critical Survey of Security Indicator Approaches." In: *2012 Seventh International Conference on Availability, Reliability and Security*. IEEE, Aug. 2012. doi: 10.1109/ares.2012.10.

- [112] P. Runeson and M. Höst. "Guidelines for conducting and reporting case study research in software engineering." In: *Empirical Software Engineering* 14.2 (2008), pp. 131–164. DOI: 10.1007/s10664-008-9102-8.
- [113] S. Ryan and R. V. O'Connor. "Acquiring and sharing tacit knowledge in software development teams: An empirical study." In: *Information and Software Technology* 55.9 (2013), pp. 1614–1624. DOI: 10.1016/j.infsof.2013.02.013.
- [114] S. Ryan and R. V. O'Connor. "Development of a team measure for tacit knowledge in software development teams." In: *Journal of Systems and Software* 82.2 (2009), pp. 229–240. DOI: 10.1016/j.jss.2008.05.037.
- [115] N. Schenk. "An Adaptive Approach for Security Compliance in Large-Scale Agile Software Development." MA thesis. Technical University Munich, 2022.
- [116] SecurityRAT. <https://securityrat.github.io/>. Apr. 2021.
- [117] S. Sengupta. *Your guide to DevSecOps Automation*. Crashtest Security. <https://crashtest-security.com/devsecops-automation/>. July 2022.
- [118] H. Setiawan, L. E. Erlangga, and I. Baskoro. "Vulnerability Analysis Using The Interactive Application Security Testing (IAST) Approach For Government X Website Applications." In: *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*. 2020, pp. 471–475. DOI: 10.1109/ICOIACT50329.2020.9332116.
- [119] M. Siavvas, D. Kehagias, D. Tzovaras, and E. Gelenbe. "A hierarchical model for quantifying software security based on static analysis alerts and software metrics." In: *Software Quality Journal* (May 2021). DOI: 10.1007/s11219-021-09555-0.
- [120] S. T. Siddiqui. "Significance of Security Metrics in Secure Software Development." In: *International Journal of Applied Information Systems* 12.6 (Sept. 2017), pp. 10–15. DOI: 10.5120/ijais2017451710.
- [121] A. SIGSOFT. *Qualitative Surveys (Interview Studies)*. <https://github.com/acmsigsoft/EmpiricalStandards/blob/master/docs/QualitativeSurveys.md>.
- [122] K. Silhoub, F. Nembhard, and M. Carvalho. "A Metrics Tracking Program for Promoting High-Quality Software Development." In: *2019 SoutheastCon*. 2019, pp. 1–8. DOI: 10.1109/SoutheastCon42311.2019.9020395.
- [123] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford. "Questions developers ask while diagnosing potential security vulnerabilities with static analysis." In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015. DOI: 10.1145/2786805.2786812.
- [124] R. van Solingen, V. Basili, G. Caldiera, and H. D. Rombach. *Goal Question Metric (GQM) Approach*. 2002. DOI: 10.1002/0471028959.sof142.
- [125] SonarQube. *Metric Definitions*. <https://docs.sonarqube.org/latest/user-guide/metric-definitions/>.
- [126] S. Springett. *Component Analysis*. OWASP. https://owasp.org/www-community/Component_Analysis. July 2022.

- [127] O. Stahl, R. Baecker, M. Bartholdt, S. Roth, and A. Sejdiu. *Cybersecurity as a Matter of Competitive Advantage*. <https://www.porsche-consulting.com/en/home/news/cybersecurity-as-a-matter-of-competitive-advantage/>. 2021.
- [128] C. Steinmann and H. Stienen. "SynQuest - Tool Support for Software Self-Assessments." In: *Software Process: Improvement and Practice 2.1* (1996), pp. 5–12. DOI: 10.1002/(sici)1099-1670(199603)2:1<5::aid-spip33>3.0.co;2-u.
- [129] V. Stray, N. B. Moe, and R. Hoda. "Autonomous agile teams." In: *Proceedings of the 19th International Conference on Agile Software Development: Companion*. ACM, May 2018. DOI: 10.1145/3234152.3234182.
- [130] M. R. Stytz and S. B. Banks. "Dynamic software security testing." In: *IEEE Security & Privacy* 4.3 (2006), pp. 77–79. DOI: 10.1109/MSP.2006.64.
- [131] K. Sultan, A. En-Nouaary, and A. Hamou-Lhadj. "Catalog of Metrics for Assessing Security Risks of Software throughout the Software Development Life Cycle." In: *2008 International Conference on Information Security and Assurance (isa 2008)*. IEEE, Apr. 2008. DOI: 10.1109/isa.2008.104.
- [132] W. Swasty and A. R. Adriyanto. "Does Color Matter on Web User Interface Design." In: *CommIT (Communication and Information Technology) Journal* 11.1 (2017), p. 17. DOI: 10.21512/commit.v11i1.2088.
- [133] Synopsys. *Seeker: Interactive Application Security Testing*. 2022.
- [134] Synopsys. *Software SecurityMetrics Development*. Synopsys. 2017.
- [135] =, and B. G. Kiliç. "Holistic Web Application Security Visualization for Multi-Project and Multi-Phase Dynamic Application Security Test Results." In: *IEEE Access* 9 (2021), pp. 25858–25884. DOI: 10.1109/ACCESS.2021.3057044.
- [136] C. Tozzi. *6 DevSecOps Metrics for DevOps and Security Teams to Share*. <https://thenewstack.io/6-devsecops-metrics-for-devops-and-security-teams-to-share/>. Sept. 2020.
- [137] F. M. Tudela, J.-R. B. Higuera, J. B. Higuera, J.-A. S. Montalvo, and M. I. Argyros. "On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications." In: *Applied Sciences* 10.24 (2020), p. 9119. DOI: 10.3390/app10249119.
- [138] O. Turetken, I. Stojanov, and J. J. M. Trienekens. "Assessing the adoption level of scaled agile development: a maturity model for Scaled Agile Framework." In: *Journal of Software: Evolution and Process* 29.6 (July 2016), e1796. DOI: 10.1002/smr.1796.
- [139] R. Velasco. *What is IAST? All About Interactive Application Security Testing*. <https://hdivsecurity.com/bornsecure/what-is-iast-interactive-application-security-testing/>. May 2020.
- [140] J. Voas and R. Kuhn. "What Happened to Software Metrics?" In: *Computer* 50.5 (May 2017), pp. 88–98. DOI: 10.1109/mc.2017.144.

- [141] T. J. Wagner and T. C. Ford. "Metrics to Meet Security - Privacy Requirements with Agile Software Development Methods in a Regulated Environment." In: *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, Feb. 2020. DOI: 10.1109/icnc47757.2020.9049681.
- [142] J. Walden. *Software Metrics*. Northern Kentucky University.
- [143] J. Walden, J. Stuckman, and R. Scandariato. "Predicting Vulnerable Components: Software Metrics vs Text Mining." In: *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, Nov. 2014. DOI: 10.1109/issre.2014.32.
- [144] J.-P. Watzelt. "Design and Implementation of a Team Maturity Model Assessing Security Compliance in Large-Scale Agile Software Development." MA thesis. Technical University Munich, 2022.
- [145] J. Webster and R. T. Watson. "Analyzing the Past to Prepare for the Future: Writing a Literature Review." In: *MIS Quarterly* 26.2 (2002), pp. xiii–xxiii. ISSN: 02767783.
- [146] C. Woody, R. Ellison, and W. Nichols. *Predicting Software Assurance Using Quality and Reliability Measures*. Tech. rep. CMU/SEI-2014-TN-026. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2014.
- [147] J. Yang, L. Tan, J. Peyton, and K. A. Duer. "Towards Better Utilizing Static Application Security Testing." In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, May 2019. DOI: 10.1109/icse-seip.2019.00014.
- [148] E. Yasasin and G. Schryen. *Requirements for IT Security Metrics - an Argumentation Theory Based Approach*. 2015. DOI: 10.18151/7217537.
- [149] U. Zdun, P.-J. Queval, G. Simhandl, R. Scandariato, S. Chakravarty, M. Jelic, and A. Jovanovic. "Microservice Security Metrics for Secure Communication, Identity Management, and Observability." In: *ACM Transactions on Software Engineering and Methodology* (May 2022). DOI: 10.1145/3532183.