From Legacy Designs to Vulnerability Fixes: Understanding SAST Adoption in Non-Technological Companies

Luis Amaral*, Michael Schlichtig†, Wagner Emanuel*, Joilton Almeida*, Carine Ferreira*, Jerome Kempf‡, Rodrigo Bonifácio*, Eric Bodden†, Laerte Peotta*, Gustavo Pinto§, and Márcio Ribeiro¶

*Department of Computer Science, University of Brasília, Brazil
†Paderborn University and Fraunhofer IEM, Germany
‡Paderborn University, Germany
§Faculty of Computing, Federal University of Pará (UFPA) and Zup Innovation, Brazil
¶Computing Institute, Federal University of Alagoas, Brazil

***Abstract*—Static analysis tools are essential for identifying software quality issues, bugs, and security vulnerabilities in source or binary code. Despite their importance, there is limited research on how non-tech companies approach software security and integrate Static Application Security Testing (SAST) tools into their development pipelines. To advance understanding in this area, we examine the motivations, challenges, and benefits of SAST tools and present an industry case study on using CogniCrypt and CryptoGuard to detect cryptographic API misuse, a common source of software vulnerabilities. We conducted focus groups with software developers and security experts from three organizations and performed a thematic analysis following a grounded theory approach to identify key factors influencing SAST tool adoption and participants' perceptions of these tools' effectiveness in detecting and reporting vulnerabilities. Based on this analysis, we observed themes reflecting concerns about false positive rates, a tendency toward overconfidence in state-of-the-practice security tools, reliance on infrastructure-based data protection measures, and persistent challenges in remediating vulnerabilities in legacy systems. Despite these challenges, participants recognized the value of SAST tools in enhancing security awareness, facilitating knowledge transfer, and contributing to secure software development practices. Based on these insights, we provide actionable recommendations to support SAST adoption and improve the usability, reporting clarity, and integration of tools such as CogniCrypt and CryptoGuard. As a concrete outcome, one of the companies decided to remediate all cryptographic API misuses that we identified in their systems during our collaboration.**

***Index Terms*—Software Security, Static Application Security Testing, Empirical Software Engineering**

I. Introduction

Software security is becoming increasingly important for developers to protect data, especially as software applications are collecting more and more sensitive user information. According to Ferguson et al., even systems designed by security experts may be broken a few years later, highlighting the fact that security in general, and cryptography in particular, are "fiendishly difficult" [1]. In the same vein, Nadi et al. concluded that software developers often struggle with Java cryptography APIs, primarily due to a lack of familiarity with software security and because these APIs are often poorly documented and difficult to use [2]. Partly for these reasons, cryptographic API misuse has been identified as a common cause of many security vulnerabilities [2]–[5]. As an example, Listings 1 and 2 illustrate a cryptographic API misuse recorded as CVE-2023-32699. In this case, the insecure MD5 hashing algorithm is used for storing user passwords in the MeterSphere project, making it susceptible to security vulnerabilities.

To alleviate this problem, practitioners might benefit from using static or dynamic analysis tools to detect source code vulnerabilities, including those caused by cryptographic API misuses. Despite the number of empirical studies on the use of static analysis tools in general [6]–[9] or specifically on detecting software vulnerabilities [10], [11], there remains limited knowledge about how software developers and security experts in non-technological companies understand and integrate SAST tools. To address this gap in the literature, here we report on a qualitative study that characterizes the adoption of SAST tools—including their motivations, benefits, and challenges—and examines how developers and security experts perceive the use of CogniCrypt and CryptoGuard (two state-of-the-art static analyzers for detecting cryptographic API misuses) to identify vulnerabilities in Java enterprise systems and libraries. Our research draws on three focus groups (FGs) conducted with Java developers and security specialists from diverse organizations: an agricultural research company, a state court, and a financial institution. Section II provides details about our study settings.

Our findings lead to several recommendations for adopting SAST tools, including prioritizing different categories of warnings, determining which components should be analyzed, and addressing common root causes of vulnerabilities (e.g., legacy design decisions, buggy code copied from the internet). We also highlight potential benefits of static analysis (e.g., facilitating knowledge transfer) and provide insights into how security experts and developers perceive the effectiveness of specialized tools such as CogniCrypt and CryptoGuard in detecting cryptographic API misuses. Conducting this research

```
public void createUser(User userRequest) {
    User user = new User();
    BeanUtils.copyProperties(userRequest, user);
    user.setCreateTime(System.currentTimeMillis());
    user.setPassword( CodingUtil.md5(user.getPassword()) );
    // ...
    userMapper.insertSelective(user);
}
```

Listing 1. User Creation with MD5 Password Hashing

```
public static String md5(String src) {
     return md5(src, UTF_8);
}

public static String md5(String src, String charset) {
    try {
        byte[] strTemp = src.getBytes(charset);
         MessageDigest md = MessageDigest.getInstance("MD5");
         md.update(strTemp);
         byte[] hash = md.digest();
        // ...
    } catch (Exception e) {
        throw new RuntimeException("MD5 encrypt error:", e);
    }
}
```

Listing 2. Insecure MD5 Hashing Utility

in non-technical companies reveals findings and recommendations not previously discussed in the literature, including:

- A common misconception is that when infrastructure policies are implemented, or when industry-ready tools such as CT01[1] and CT02 do not report any security vulnerabilities, a system can be considered secure.
- Decisions that were considered correct in the past might have led to known vulnerabilities in legacy systems, and these are difficult to fix due to a lack of understanding of how changes will propagate through different components and systems.
- Recommendations for improving tools like CogniCrypt and CryptoGuard include the need to integrate them into continuous integration pipelines and reduce false positives by excluding warnings from dead or unreachable code.

II. Study Settings

This research has two main goals. First, it aims to understand the motivations, benefits, and challenges that non-technology companies face when integrating static application security testing (SAST) tools into their development workflows. Second, it investigates how software developers and security experts from these companies respond to warnings about cryptographic API misuses—specifically, those reported by CogniCrypt and CryptoGuard, two state-of-the-art static analysis tools for detecting Java Crypto API misuses.

We focus on cryptographic API misuses because this class of vulnerabilities is recognized among the OWASP Top Ten. Although not a primary objective, an important side effect of this research is to provide designers of misuse-detection tools, such as CogniCrypt and CryptoGuard, with actionable insights that may enhance their effectiveness and foster their broader adoption. Since both tools operate on languages compiled to Java bytecode, our study targets security experts and software developers who work on Java-based enterprise systems developed internally within non-technology companies—i.e., organizations that build in-house systems to support their core business operations. In this way, our findings complement prior research, which has primarily examined perceptions within technology companies such as Microsoft and Google (e.g., [12], [13]).

For data collection, we relied on both the outputs of static analysis tools and the focus group (FG) method [14] to derive *organization-level perceptions*, rather than individual viewpoints, from practitioners [15]. We engaged developers contributing to Java enterprise systems across diverse domains. The study yielded a rich set of insights—ranging from how organizations integrate SAST tools into their development workflows to how security experts and developers interpret and prioritize crypto-API misuse warnings.

For data synthesis, we adopted the Socio-Technical Grounded Theory (STGT) method [16], [17] to consolidate the focus-group results into a set of hypotheses. Consistent with previous studies [18], [19], we used STGT solely for qualitative data analysis rather than for constructing a full-fledged theory [17]. We made this methodological choice because our investigation centers on the perspectives of software developers and security experts from three distinct companies. Although these organizations exhibit different levels of experience with SAST tools, we do not claim to have reached data saturation (the mains reason for not proposing a theory).

STGT builds upon procedures adapted from the Glaserian [20], Strauss–Corbinian [21], and Constructivist [22] variants of traditional Grounded Theory (GT). The method enables us to derive organization-specific knowledge from software practitioners and security experts by examining both their experiences with SAST tools and the outputs of crypto-API misuse detectors, thereby grounding our findings in empirical evidence. The following subsections provide further details about the study participants and the procedures used for data collection and synthesis.

A. Participant Selection

We adopted a convenience sampling strategy to recruit participants for our study. Using our professional network, we contacted software developers and security experts who were interested in sharing their experiences with SAST tools in their organizations. Developers from three companies (**Company A**, **Company B**, and **Company C**), each representing a distinct organizational context, agreed to participate. **Company A**

[1]The organizations in our study mentioned two commercial tools. We checked with the vendors and upon request omitted the names of the commercial tools - using the pseudonyms CT01 and CT02. This decision further ensures that our analysis remains impartial and is not perceived as endorsing or criticizing specific commercial tools.

is a large agricultural research corporation with approximately 10,000 employees, including more than 2,400 researchers. **Company B** is a judiciary institution from the Brazilian Federal District that manages information systems to support its judicial and administrative activities. **Company C** is one of the largest banks in Latin America, employing around 87,000 people, of whom approximately 6,000 work in the IT department. The IT complexity of these organizations varies considerably, as they maintain from dozens to hundreds of enterprise systems.

Each company participated in a dedicated focus group session. The composition of these sessions varied according to the professional profiles available in each organization: **Company A** included only software developers, **Company B** involved software developers and technical leads, and **Company C** comprised software developers and security specialists. This cross-company variation provided a heterogeneous set of perspectives on SAST adoption, reflecting both development and security viewpoints. Participants also differed in their familiarity with SAST tools, ranging from limited exposure to routine use, which enabled us to capture a broad spectrum of experiences. In total, 16 practitioners participated across the three focus group sessions (see Table I).

Most participants were software developers, while three served as technical leads and another three specialized in cybersecurity (all from **Company C**). Their participation was motivated not only by their involvement in secure software development practices but also by the opportunity to experiment with CogniCrypt and CryptoGuard in their own development contexts. Several participants viewed the study as a chance to identify potential cryptographic vulnerabilities in their systems that had not been previously detected by state-of-the-practice static analysis tools.

TABLE I
DEMOGRAPHICS OF FOCUS GROUP PARTICIPANTS. ROLES INCLUDE SOFTWARE DEVELOPER (SD), TECHNICAL LEADER (TL), AND CYBERSECURITY EXPERT (CE). EXPERIENCE IS REPORTED IN YEARS. THE SAST TOOLS COLUMN INDICATES THE TOOLS THE PARTICIPANTS ARE FAMILIAR WITH.

Practitioner	Company	Experience	Role	SAST Tools
PA1	A	20 years	SD	CT01
PA2	A	22 years	SD	None
PA3	A	20 years	SD	CT01
PA4	A	17 years	SD	CT01
PA5	A	26 years	SD	None
PA6	A	15 years	SD	CT01
PB1	B	20 years	SD	CT01
PB2	B	15 years	TL	CT01
PB3	B	30 years	SD	CT01
PB4	B	21 years	TL	CT01
PC1	C	6 years	SD	CT01
PC2	C	12 years	SD	CT01
PC3	C	4 years	CE	CT02
PC4	C	10 years	CE	CT01/CT02
PC5	C	15 years	TL	CT01/CT02
PC6	C	17 years	CE	CT01/CT02

B. Data Collection Procedures

We conducted three distinct focus group sessions, one for each participating company. As part of the preparation, developers from the respective organizations were asked to run CogniCrypt and CryptoGuard on a sample of Java projects they develop or maintain. Participants were subsequently asked to share the outputs produced by these tools with our research team. We scheduled the focus group sessions only after thoroughly analyzing the cryptographic API warnings generated by CogniCrypt and CryptoGuard on the selected systems, which allowed us to prepare targeted discussion materials for each session. This setup enabled us to explore not only their broader experience integrating program analysis tools into their workflows but also their experience using CogniCrypt and CryptoGuard. Exploring concrete issues found in the participants' systems is a distinguishing feature of our study.

We structured the sessions into three phases. In the first phase, we gathered demographic information, including participants' experience with software development and software security, their roles within their companies, and their familiarity with program analysis tools—particularly SAST tools. We also examined the projects selected by participants, noting the project domain, number of contributors, and complexity (in terms of lines of code). In the second phase, we explored how program analysis tools are used within each company's development workflow, along with participants' perceptions of the tools they were already familiar with (see Table I). Finally, in the third phase, we focused on participants' perceptions of the vulnerabilities detected by CogniCrypt and CryptoGuard for the analyzed systems. All focus group sessions were recorded.

Each phase followed a semi-structured interview protocol, allowing for on-the-fly elaboration and discussion. This approach encouraged participants to provide detailed responses to context-dependent questions. Below, we illustrate some of the questions that emmerged during the third phase:

- Could the use of the MD5 hash function in this context pose a potential security threat?
- From your perspective, is it crucial for your team to fix this issue? Please elaborate on your decision-making process.
- To what extent do the warning messages generated by these tools provide clarity? Are there any essential details that you believe are missing?
- What suggestions would you make to improve the effectiveness and overall quality of these tools?

The first focus group involved practitioners from **Company A**. During the preparation phase, practitioners from **Company A** selected sixteen Java components (systems or libraries) and executed analyses with CogniCrypt and CryptoGuard. Both tools detected crypto-API misuses in six components. These included a library supporting authentication mechanisms using LDAP (Lightweight Directory Access Protocol) and AD (Windows Active Directory), and a reference architecture implementation for Java Enterprise Edition (JEE) systems.

The second focus group was conducted with practitioners from **Company B**. They selected six Java systems across different domains for analysis with CogniCrypt and CryptoGuard. Both tools reported crypto API misuses in only one system—a court precatory system developed with Java Enterprise Edition (JEE) and JavaServer Faces (JSF). This system was originally designed by another institution and later incorporated by **Company B**. Developers noted that they already use the commercial tool CT01 to detect software vulnerabilities and code smells, and therefore did not expect CogniCrypt and CryptoGuard to uncover additional issues.

The third focus group involved practitioners and security specialists from **Company C**. We spent two weeks at the **Company C** headquarters working closely with security specialists to understand their experience integrating SAST tools with continuous integration (CI) workflows. Practitioners then selected 48 Java systems and libraries for analysis, enabling us to discuss the results of CogniCrypt and CryptoGuard. Out of the 48 Java components analyzed, CogniCrypt flagged warnings in 15, while CryptoGuard identified misuses in 23. We focused on a representative subset of these warnings during the group discussion to capture participants' insights and perceptions.

After concluding all sessions, we transcribed the recordings and organized the resulting data using the Atlas.TI tool. Due to confidentiality constraints, only part of the data is publicly available in our replication package.[2]

C. Data Analysis Procedures

To analyze the collected data, we conducted a qualitative analysis grounded in the principles of STGT, involving interleaved activities between *open coding* and *constant comparison*, followed by *theoretical modeling*. In particular, *open coding* and *constant comparison* were carried out in parallel, as described below.

Open Coding. In this stage, we systematically annotated relevant excerpts from the transcripts with emerging codes. Through a process of constant comparison, we continuously examined and refined these codes to relate them to broader conceptual themes. The main goal was to iteratively identify, refine, and group codes into higher-level concepts. We also recorded memos—that is, concise comments capturing the essence of each statement. For example, we annotated the following excerpt:

> "[...] the warnings these tools presented are now seen as potential security flaws. However, at the time when these components were developed, they were considered up-to-date and believed to be secure."

with the code **secure flaws come from legacy decisions** and the memo *Design decisions that were once secure may later lead to and propagage vulnerabilities, especially when components are reused across systems.* In several cases, we assigned multiple codes to the same excerpt. Three authors independently performed this step, iteratively reviewing and refining the coding until reaching consensus.

Constant Comparison. Conducted in parallel with open coding, this activity involved grouping conceptually related codes into broader concepts and categories. For instance, excerpts annotated with "*Change Impact Assessment*" and "*Change Propagation*" were merged into the higher-level category *CAT05 – Impact Assessment to Fix a Vulnerability*. This process required iterative discussions among the authors to ensure conceptual coherence and category consolidation. The resulting categories are shown in Table II.

Theoretical Coding. Following Grounded Theory principles, we engaged in theoretical coding to integrate the categories identified in the previous stages into a cohesive conceptual model. This model summarizes the core hypotheses that emerged from our analysis and delineates the relationships among the main categories. As shown in Table II, six categories were identified, all centered around the core concept *Adoption of SAST Tools in Non-technological Companies*. We refined these hypotheses through several rounds of discussion until consensus was reached among at least three authors. As noted in the previous section, the resulting model does not constitute a fully developed theory but rather a consolidated set of hypotheses that explain the relationships among categories.

TABLE II
MAIN CATEGORIES THAT EMERGED FROM OUR STUDY.

Category ID	Category Name
CAT01	Benefits of Using SAST Tools
CAT02	Limitations and Challenges of Using SAST Tools
CAT03	Root Causes of Vulnerabilities
CAT04	SAST Tools Adoption
CAT05	Impact Assessment to Fix a Vulnerability
CAT06	General Overconfidence

Our research on *SAST tool adoption in non-technological companies* spans a wide range of organizational contexts, from a company with almost no prior experience using static analysis (**Company A**) to another that makes substantial investments in both SAST and DAST tools to detect software vulnerabilities (**Company C**). The next section summarizes the core observations derived from each company, while Section IV discusses the main categories that emerged in our study (see Table II) and highlights relevant quotes from the transcripts.

III. CROSS-COMPANY SUMMARY OF OBSSERVATIONS

This section brings together the core observations from the three focus groups, outlining shared concerns and responses to the vulnerabilities reported by CogniCrypt and CryptoGuard.

A. Summary of Observations at Company A

Discussions at **Company A** indicated that software security was not yet a primary focus in the participants' development practices. Their priorities were primarily centered on functional requirements, reflecting a belief that developers should rely on the underlying infrastructure to prevent vulnerabilities

[2]https://github.com/luisamaralh/saner26-supplementary-material

from being exploited by malicious actors. This assumption overlooks, for instance, that some attacks may originate from insiders. Participants also described a frustrating experience when attempting to integrate a commercial tool into their development pipeline, mainly because it significantly slowed the build process and frequently caused build failures.

After we presented the cryptographic API misuses reported by CogniCrypt and CryptoGuard, participants engaged in a long discussion about whether the vulnerable code was actually in use. This reaction suggests that these tools should provide richer contextual information to help developers determine where vulnerable code is invoked. Understanding the problem, deciding how to implement fixes, and anticipating how those fixes might propagate to other components emerged as central concerns during the focus group session at **Company A**. The participants acknowledged that such tools could play a constructive role in raising awareness of security issues, highlighting the potential of SASTs not only to detect vulnerabilities but also to knowledge transfer.

Participants also expressed a willingness to estimate the effort required to address the reported warnings and recognized that applications exposing external interfaces should be prioritized for security assessment—reflecting an understanding of risk-based prioritization and an initial commitment to remediating some of the vulnerabilities reported by CogniCrypt and CryptoGuard.

B. Summary of Observations at Company B

The use of static analysis tools such as CT01 had long been institutionalized within the company. Developers employed this tool not only to enhance software security but also to improve software quality more broadly. At the time of the focus group, participants described an ongoing initiative aimed at resolving all vulnerabilities reported by CT01 in their enterprise systems. They also expressed strong confidence in the tool, asserting that the analyzed systems were "zeroed" with respect to security vulnerabilities. Furthermore, their deployment pipeline was configured to block production releases whenever unresolved vulnerability issues were detected.

Consistent with findings from the focus group at **Company A**, participants devoted considerable effort to assessing the severity of the warnings produced by CogniCrypt and CryptoGuard, even while acknowledging the presence of false positives. For example, CogniCrypt flagged a vulnerability related to hardcoded credentials, which participants dismissed as a false positive since, in production, those credentials are stored in protected configuration files. Once again, several vulnerabilities were reported in utility classes that were no longer in use—highlighting the importance of conducting periodic static analysis audits to identify and remove inactive yet vulnerable code. After discussing the outcomes produced by CogniCrypt and CryptoGuard, participants also noted that tools of this kind could assist in decisions about whether to incorporate systems developed by other companies. Overall, the depth of discussions at Company B suggests a more mature security awareness compared to Company A, where learning processes were still emerging.

A recurrent finding regarding false positives was that the tools often reported vulnerabilities that could only manifest in the development environment, but not in production. This observation suggests that static analysis tools should be able to differentiate between system configurations and execution contexts (e.g., development vs. production) during analysis. Participants also reported that the configuration of CogniCrypt and CryptoGuard was not straightforward and suggested that such tools should be available as plugins for platforms already integrated into their pipelines, such as CT01. Despite these difficulties, they recognized the more specialized and thorough analyses provided by CogniCrypt and CryptoGuard, expressing surprise that these tools detected vulnerabilities overlooked by CT01. Finally, participants mentioned scalability issues when running these tools on large projects at **Company B**.

C. Summary of Observations at Company C

In **Company C**, the use of multiple SAST and DAST tools is fully institutionalized. The organization developed a research effort to propose a quality score, and any program version failing to meet it cannot advance through their development pipeline. That is, participants at **Company C** expressed high confidence in the tools' outputs and in the statistically defined score threshold.

After reviewing vulnerabilities reported by CogniCrypt and CryptoGuard, discussions moved toward deeper technical considerations. Some vulnerable code dated back more than ten years, to a period when the underlying design decisions were considered secure. Although some reported vulnerabilities were deemed unlikely to be exploitable in practice, participants acknowledged the need to revisit older design choices periodically, as insecure cryptographic primitives persist in the code base.

Participants also noted that not all developers would fully understand the impact of certain changes necessary to fix a vulnerability—such as increasing a message size from 160 bits to 256 bits—which, although simple to implement, may disrupt inter-system communication or affect user authentication. As a result, careful change-impact analysis is required before remediating vulnerabilities. They also expressed concern about fixing vulnerabilities in components that are reused across different systems. We also observed that **Company C** also faces cultural and operational challenges. A dedicated software security team reviews selected warnings to reduce false positives, yet false-positive rates remain high (around 60%). Participants emphasized the need to *strike a balance* between improving security and sustaining developer productivity, especially when teams must reconcile large volumes of warnings with the pressure to deliver new features. Finally, they suggested that CogniCrypt and CryptoGuard should provide more detailed warning messages to better support developers' decision-making.

IV. Key Categories

The results of our data analysis using STGT practices led to six key categories, ranging from the benefits and challenges of using SAST tools to instances of overconfidence in their outcomes. In this section, we summarize these categories, placing greater emphasis on the themes that emerged most frequently.

A. Benefits of Using SAST Tools

Benefits such as improved software quality, consistency, and vulnerability detection have already been discussed in the literature on static analysis tool adoption [6], [23], [24]. These benefits also emerged in our study. Nonetheless, participants in our focus groups identified additional advantages that are not frequently emphasized in prior work.

For instance, participants from **Company A** and **Company C** highlighted the importance of static analysis tools for enhancing security awareness and supporting knowledge transfer within development teams. As one participant from **Company A** explained: **(PA2)** "*These tools might help us become more familiar with security issues, which are sometimes overlooked. These tools gradually reveal where the errors occur, right? So, I believe it's a good practice for us to reconsider our development process and add that security layer.*"

Participants from **Company B** also emphasized that static analysis tools (such as CT01) are essential not only for improving system quality but also for informing decisions about whether to adopt systems developed by external organizations. In their context, this is particularly relevant because **Company B** frequently reuses software produced by other judiciary agencies. During the focus group, participants described how static analysis tools help them assess the feasibility and risks associated with such reuse.

As one participant noted: **(PB1)** "*Taking advantage of this discussion, it would be interesting for us, when we analyze the feasibility of using a system provided by another agency, [. . .], we could diagnose these systems with these tools and use our background with* CT01 *to say: 'we will not incorporate this system because it is bugged'. Or if not, 'this system is healthy and only needs small adjustments'.*"

B. Limitations and Challenges of Using SAST Tools

Participants at **Company A** had a frustrating experience using CT01, which eventually led them to remove static analysis tools from their development pipeline. In particular, after attempting to integrate a commercial tool, they observed significant slowdowns in build time and unexplained build failures that prevented execution from proceeding. These issues may have been caused by a misconfiguration of the tool. Participants also noted that, while there is abundant literature on software testing, the literature on static analysis is comparatively scarce. As a result, they reported not fully understanding the potential benefits of using static analysis tools. Consequently, they tend to invest more in testing than in other techniques aimed at improving software quality.

False positives are a recurrent limitation in the literature on the adoption of static analysis tools (we refer the reader to [25] for an up-to-date overview of this topic). We also observed this concern in our study, but the focus groups revealed more specific causes of false positives that were not necessarily related to conservative approximations in the analyses [25]. For instance, during the focus group at **Company A**, several participants questioned whether the crypto API misuses reported by CogniCrypt and CryptoGuard could actually be exploited in practice. This doubt emerged because some warnings originated from unused code, testing code, or from the use of `java.util.Random` in non-sensitive contexts, which they perceived as false positives.

Similarly, participants from **Company B** identified situations where CogniCrypt and CryptoGuard reported warnings they considered false positives. For example, CogniCrypt issued a warning indicating that a parameter used to configure an instance of `java.security.KeyStore` should not be hardcoded. After analyzing the case, one participant explained: **(PB4)** "*[. . .] from what I see here, the implementation fetches everything from a system configuration file. The KeyStore parameter and the password are within the system configuration file. So this information is not hardcoded.*" Participants also noted that some warnings were triggered by deprecated methods that are not used in production, further contributing to their perception of irrelevant warnings.

Security experts from **Company C** also expressed concerns about the high rate of false positives generated by static analysis tools more generally (i.e., not only CogniCrypt and CryptoGuard), emphasizing their potential to reduce developers' reliance on these tools and slow down development processes. According to the participants, one challenge is determining which vulnerabilities truly matter in specific organizational contexts. False positives can require considerable effort from development teams to triage, reducing their efficiency. As one participant stated: **(PC5)** "*We're also working on prioritizing and refining security rules and policies because we know that initially when these tools are integrated, they generate around 60% false positives. So, we're analyzing what's happening out there so that we can prioritize rules for managing the security of the SAST process.*"

To mitigate these issues, participants from **Company C** described a proactive approach: **(PC3)** "*We review the results (of static analysis tools) before sending them to the development teams, to remove the number of false positives, which is often a very high number. So we do this work to spare the developer from trying to solve something that is actually a false positive.*" These efforts aim to ensure that static analysis tools deliver actionable results, fostering communication between developers and security experts, and aligning with the broader principles of DevSecOps [26]. **(PC5)** "*Our team here is responsible for managing the entire process. Our process involves communication with development teams to find an optimal balance between development and security in the pipeline.*"

Another limitation raised by participants concerns the lack

of guidance on how to fix certain issues, especially in cases where developers have limited experience with specific libraries or APIs. For example, developers from **Company A** reported difficulties interpreting warnings related to the JCA library: **(PA1)** "*[. . .] (for this particular case, to fix the misuses), would it be necessary to develop new code, or just change the cipher configurations (from DES to something else)? [. . .] Okay. Sounds good, we'll look into it later.*" Participants from **Company C** similarly noted that, although the tools effectively identify issues, the accompanying messages are sometimes generic, forcing developers to leave their workflow to search for additional information: **(PC3)** "*Sometimes the information these tools provide is a bit generic, we realize that, so the developer has to leave the environment to research and try to understand a way to correct it.*"

The clarity and level of detail in warning messages were also discussed. When reviewing warnings from CogniCrypt and CryptoGuard related to the misuse of a message digester implementation, one participant observed: **(PC3)** "*If we consider the number of people we have in the company very few [developers] would know which of these three options [SHA-256, SHA-384, SHA-512] to choose (to fix the problem), considering performance and processing time.*"

Finally, scalability limitations were also mentioned. One participant from **Company B** reported that both CogniCrypt and CryptoGuard were unable to complete the analysis of a complex system: **(PB1)** "*[. . .] the tool couldn't finish the execution and generation of the report for a complex system (a very big one at our company)*".

C. Root Causes of Vulnerabilities

Participants from **Company B** suggested that some of the vulnerabilities reported by CogniCrypt and CryptoGuard may have originated from code copied and pasted from the web, supporting findings from previous work [2], [27]–[29]. In our study, participants frequently mentioned Q&A sites and code assistants as primary sources to learn how to (mis)use cryptographic APIs. At **Company B**, developers also copy and paste code from industrial patterns. As one participant described: **(PB4)** "*[. . .] the probability that this code was copied and pasted from somewhere is huge. From what I'm seeing here (a piece of code with a crypto API misuse), it's very likely that someone just copied and pasted from some other code.*" This observation ultimately motivated development teams at **Company B** to integrate CT01 into their development pipelines. **(PB1)** "*[. . .] this matches very well with part of the study we conducted here and that even motivated the use of these static analysis tools. A task often comes to a developer like: you have to develop a Java feature that must calculate the hash of some data and then perform a functionality with the resulting hash. The developers then go to Stack Overflow and copy a piece of code as is. The feature works fine and then he/she leaves it like that.*"

Participants from **Company A** recognized that some of the vulnerabilities reported by CogniCrypt and CryptoGuard may have been introduced into architectural components more than ten years ago. At the time, those cryptographic design decisions might even have been considered adequate. However, due to the difficulties of modernizing legacy code and the extensive dependencies that various systems have on these components, updating them has become a challenging, risky, and costly task—one that technical teams would have little inclination to undertake. A similar perspective about legacy code also emerged during the focus group at **Company C**. We found that legacy code often reflects implementation decisions made under outdated cryptographic standards and knowledge available at the time. These decisions, while appropriate in their original context, now contribute to several of the misuses reported by CogniCrypt and CryptoGuard.

That is, what was once an adequate solution may later conflict with current standards and evolving vulnerabilities. As one participant explained: **(PC6)** "*This code must be around 13 years old. The choice of SHA1 at the time was to avoid using MD5, which would be a one-way function of 128 bits. At that time, SHA1 was robust [. . .]. Therefore, we didn't update the code to the current recommendations, and I even consider that we should review these codes from time to time. In fact, DES is still used in some systems, and today it is easily broken.*" Legacy systems and their dependencies surfaced as significant barriers to implementing changes suggested by SAST tools. Other root causes of vulnerabilities at **Company A** relate to (a) limited software security knowledge, (b) an over-reliance on the underlying infrastructure, and (c) security being treated as a secondary concern. We also observed that the prioritization of only critical vulnerabilities contributes to the persistence of non-critical issues, in an effort to maintain development agility.

D. SAST Tools Adoption

Ideas on how to integrate SAST tools emerged across the three focus groups, revealing distinct priorities and challenges in each organization. At **Company A**, participants expressed interest in adopting SAST tools in the future, initially targeting systems that expose external interfaces. Given past experiences in which these tools significantly slowed the build process, they viewed periodic assessments—rather than continuous integration checks—as more feasible.

At **Company B**, participants stressed the importance of integrating tools such as CogniCrypt and CryptoGuard into the existing analysis infrastructure already institutionalized within the company. They described difficulties configuring these tools correctly and emphasized that full adoption would require seamless integration with CT01. As one participant observed, **(PB1)** "*[. . .] to institutionalize a tool such as these (CogniCrypt and CryptoGuard) it would have to be integrated with* CT01 *[. . .]. Today we work with continuous integration [. . .], so it would be interesting to automate these analysis tools to execute at the time the build occurs, but . . . this configuration issue is not that simple.*"

Discussions at **Company C** highlighted a broader cultural dimension of SAST adoption. Participants emphasized the need for a proactive development culture aligned with DevSec-

Ops principles, in which developers are empowered to address security warnings autonomously. Knowledge transfer was seen as critical to this shift, reducing the need for developers to search externally for remediation guidance. As one participant explained, **(PC3)** "*Our main job is to facilitate developers regarding the use of tools and the process . . . bringing knowledge to the developer so that they have the necessary understanding to implement fixes quickly.*"

Participants from **Company C** also emphasized prioritizing issues based on severity and business impact. Non-blocking issues are frequently overlooked in agile teams with tight deadlines, and tools such as CT01 and CT02 help guide prioritization through their scoring mechanisms. Beyond tool outputs, participants stressed the importance of performing risk analyses when deciding whether to postpone feature delivery to address low-impact vulnerabilities. **(PC2)** "*We always have to assess the risk, whether it's something I really have to deliver today and/or what the risk is of deploying this application with a particular known vulnerability.*"

Finally, participants reported using statistical analyses and gray literature to refine tool configurations and reduce false positives, further illustrating the effort required to integrate SAST tools without creating friction in development workflows.

E. Impact Assessment to Fix a Vulnerability

A recurring concern in our study was the magnitude of changes required to fix certain vulnerabilities. During the focus group at **Company A**, participants expressed apprehension about the ripple effects of modifying an architectural component. As one participant noted, **(PA3)** "*[. . .] if you want to update the cryptography architectural component, you'd also need to update other dependencies, including the Java version we use here. It's a snowball effect where one problem leads to another, and before you know it, you're dealing with a significantly outdated environment.*"

zThis concern became evident when discussing a vulnerability reported by CogniCrypt and CryptoGuard in a library used to encrypt external users' passwords. The issue stemmed from instantiating an AES cipher in ECB mode, a long-discouraged practice associated with CWE-327.[3] The warning triggered debate about which systems actually reuse the affected component. However, the SAST reports only indicate the presence of a misuse and do not reveal whether the vulnerable method is reachable from system entry points.

Participants eventually reasoned that although many systems import the internal cryptographic library, only a few actively invoke the vulnerable functionality. This reduced the perceived impact of the fix and encouraged the team to proceed. As one developer explained: **(PA4)** "*[. . .] the library is imported in this system, but that code is not used. In fact, only one specific system uses it . . . These tools point out issues in library code that are not actually used by this program. Do they get to that level of reporting?*"

[3]See CWE-327: Use of a Broken or Risky Cryptographic Algorithm, https://cwe.mitre.org/data/definitions/327.html.

Participants from **Company C** also highlighted the broader challenge of modifying shared cryptographic utilities. Changes to such components often require coordination across multiple systems, creating significant ripple effects. As one participant observed, **(PC6)** "*If I make changes here [. . .] I have to coordinate with other systems that might use this cryptographic library, which could have a significant impact.*" In particular, the mix of technologies in use—including mainframes—may impose strict constraints, such as relying on a specific number of bytes in hash data. As explained: **(PC6)** "*It's not enough to just make the change from a 160 bits algorithm to a 256 bits one if the other side is not speaking the same language . . . a change like this can directly impact other systems that use this class.*"

F. General Overconfidence

Different forms of *overconfidence* emerged during the focus groups. At **Company A**, participants acknowledged that developers display excessive confidence in the infrastructure and operations teams, assuming that firewalls, VPNs, and server configurations sufficiently mitigate risks. As one participant noted, **(PA1)** "*Our goal is to deliver what the client requested. So, other concerns (like security) might be neglected. We believe that security should be delegated to the application server configuration, to the VPN, . . . , and ensured by the infrastructure. We only focus on feature development.*" This reliance on infrastructure leads to a limited appreciation of how vulnerabilities in architectural or reusable components can propagate across multiple systems—an issue reflected in the findings of CogniCrypt and CryptoGuard at **Company A**.

At **Company B**, a different manifestation of overconfidence was observed. Participants rely heavily on CT01, which they have increasingly used to improve code quality and address vulnerabilities. **(PB4)** "*We started using* CT01 *more heavily about two years ago. Last year we started to heavily tackle the issue of vulnerabilities and hotspots that* CT01 *indicates.*" Because their deployment pipeline blocks systems with any unresolved CT01 warnings, the team assumed their systems were free of vulnerabilities. They were therefore surprised when CogniCrypt and CryptoGuard reported an issue that CT01 had missed: **(PB4)** "*Note, this (warning that) CogniCrypt caught,* CT01 *did not point out to us. According to the* CT01 *reports, (name of the system) is zeroed in terms of vulnerabilities.*" This experience demonstrated to them that relying exclusively on CT01 may give a false sense of completeness. **(PB4)** "*We rely a lot on the outputs of* CT01 *. . . But it's interesting to know that we have other tools that further analyze our systems.*" In the end, overconfidence may lead security experts and developers to overlook other review practices, allowing vulnerabilities to propagate into production.

Practitioners at **Company C** also show strong confidence in static analysis results. Deployment to production is blocked if a system does not meet a minimum quality score defined by CT01, which gives developers assurance that severe issues will not reach production. **(PC2)** "*(The score* CT01 *assigns to*

a system) determines who can approve it for deployment. This gives us more confidence because when developers are about to deploy a system with a low score, . . . you will need to fix the system and improve its quality." Once the expected score is achieved, deployments can proceed without managerial approval. **(PC4)** "*If the* CT01 *score is good, the build and deployment can be authorized by a colleague, and it doesn't need to go to the manager.* CT01 *has these coverage and error scores that make the developer's life easier here.*"

V. A Theoretical Model of SAST Adoption in Non-technological Companies

In this section we present a theoretical model expressed through hypotheses that emerged from our research, establishing relationships among the core categories. These hypotheses are presented in Section V-A, followed by a discussion of the initial implications of our findings in Section V-B. Finally, Section V-C outlines the limitations of our study that may threaten the validity of our results.

A. Emergent Hypotheses

(H1) The double-edged sword of SAST tools. Based on CAT01 (Benefits of Using SAST Tools) and CAT06 (General Overconfidence), we found that although SAST tools facilitate vulnerability detection and knowledge transfer, developers tend to overestimate the completeness of tool reports. This overconfidence reduces the perceived need to complement SAST tools with additional security practices, thereby weakening the potential impact of the benefits identified in CAT01.

(H2) Legacy design decisions amplify the challenges of vulnerability remediation. Grounded in CAT03 (Root Causes of Vulnerabilities) and CAT05 (Impact Assessment to Fix a Vulnerability), our results show that cryptographic design decisions that were once correct may become insecure over time. This issue was a recurrent source of vulnerabilities (CAT03) across the three companies studied, compounded by the widespread practice of copying and pasting insecure code and by limited software security expertise. Legacy design choices can hinder remediation efforts (CAT05), as necessary changes propagate through multiple components in ways developers cannot easily anticipate. These findings highlight the importance of documenting and sharing experiences on modernizing legacy systems to address security vulnerabilities. To the best of our knowledge, the literature remains scarce on this topic.

(H3) Effective SAST adoption depends on configuration effort and pipeline integration. Drawing on CAT02 (Limitations and Challenges of Using SAST Tools) and CAT04 (SAST Tools Adoption), our findings show that participants either had poor experiences integrating a static analysis tool into their development pipelines—likely due to configuration challenges—or spent substantial effort fine-tuning thresholds to report only vulnerabilities that really matter. Overall, the effectiveness of static analysis tools strongly depends on proper configuration and integration into development workflows. Also, state-of-the-art tools such as CogniCrypt and CryptoGuard should be enriched with contextual information to avoid reporting warnings related to unreachable code or issues observable only in development environments.

(H4) Actionable remediation guidance strengthens the learning and decision-making process. Based on evidence from CAT02 (Limitations and Challenges) and CAT01 (Benefits of Using SAST Tools), we found that unclear or underspecified warning messages hinder developers' ability to understand the practical impact of vulnerabilities. Providing threat models, exploitation feasibility, and concrete remediation steps would enhance developers' decision-making and increase the organizational value of SAST tool adoption.

Less recurrent hypothesis Additional but less frequent patterns emerged, suggesting that: (i) scalability limitations of tools such as CogniCrypt and CryptoGuard affect adoption in large systems (CAT02), and (ii) the ripple effects of seemingly simple fixes reinforce the structural role of CAT05 (Impact Assessment) in shaping adoption decisions. These observations inform directions for further investigation but do not form core hypotheses due to their lower recurrence.

B. Implications of our Research

Our research has already yielded two concrete implications. First, developers from **Company A** launched an initiative to analyze and fix all vulnerabilities reported by CogniCrypt and CryptoGuard, aiming to foster a security-oriented mindset and assess the feasibility of remediation—especially in internal libraries that propagate vulnerabilities across different systems. This effort confirmed that all 67 vulnerabilities found in legacy systems could be manually resolved by modifying 1,024 lines of code across 12 Java classes. The fixes, however, introduced incompatibilities with legacy code, prompting the design of a migration strategy that allowed legacy and revised code to run concurrently. This approach enabled the decryption of old data and its re-encryption under the new secure standard. An industry track paper detailing this experience was published elsewhere [30].

Second, some authors of this paper, who also contribute to the CogniCrypt project, evolved the tool based on insights obtained from this study. As one of the participants pointed out that providing CWE IDs is advantageous, this has been added to CogniCrypt[4] (cf. Section IV-B). Further, insights of this study were used in a project group[5] to build a new IDE integration for CogniCrypt. **PB1** (cf. Section IV-D) explained that *workflow integration* with esisting and used commercial tools is important for CogniCrypt to be used, the project build its new user interface as plugin for a commercial tool. The project focused on many aspects also confirmed in this study, such as providing better explanations of *warning messages* and also providing*fix support* as systematically discussed by Nachtigall et al. [31]. These limitations were for instance discussed by **PA1**, **PC3**, and **PC4** (cf. Section IV-B). The developed plugin is currently under submission elsewhere.

[4]https://github.com/CROSSINGTUD/CryptSL/pull/176

[5]https://go.upb.de/pg-sse-SecAI

C. Threats to Validity

We recognize potential threats to validity that may affect the generalizability and reliability of our findings. Consdering **internal validity**, our study may be influenced by biases during the focus group discussions and data analysis. For instance, the coding process inherently depends on researchers' interpretations. To address possible threat, multiple authors independently reviewed and reconciled codes and categories, ensuring a consensus-driven approach to analysis.

Regarding **external validity threats**, the generalizability of our findings is constrained by the focus groups' participant pool, which consisted of software practitioners and security specialists from three organizations across distinct sectors. These organizations—an agricultural research company, a financial institution, and a judiciary agency—offer diversity but may not represent the broader software development community. Companies with different levels of security maturity or those using languages other than Java might face unique challenges and opportunities not captured in this study.

Some threats to **reliability validity** stem from the semi-structured nature of the focus group discussions. While this approach facilitated rich, context-specific insights, the variability in question phrasing and participant engagement across sessions may have introduced inconsistencies. To mitigate this, we followed a standardized protocol for focus groups and ensured that discussions were systematically transcribed and coded.

VI. Related Work

A substantial body of research examines how developers interact with static analysis tools, both broadly and in security contexts. Smith et al. [7], [9] conducted an observational study of how developers diagnose vulnerability warnings reported by Find Security Bugs, characterizing the questions they ask and the reasoning strategies they use. Our study complements this work by emphasizing organizational—rather than individual—factors shaping SAST adoption.

Usability has been another central focus of prior research. Smith et al. [10] evaluate several security-focused static analysis tools, finding that unclear messages and limited fix guidance hinder developers' understanding—issues also noted in our focus groups with CogniCrypt and CryptoGuard. Nachtigall et al. [31] assess 46 tools against 36 usability criteria and report pervasive weaknesses in error messages, fix support, and workflow integration. These systemic issues mirror several concerns raised in our study.

Research in "big-tech companies" has also explored static analysis adoption at scale. Sadowski et al. [13] describe Google's experience in building and deploying static analysis tools, emphasizing workflow integration and high-quality diagnostics. While Google engineers report distrust in tool outputs, we observed the opposite pattern: overconfidence in commercial SAST reports. Both studies, however, underscore the need for contextualized feedback. Similarly, Christakis and Bird [12] survey 375 Microsoft developers and find strong demand for better configurability, fewer false positives, and clearer diagnostics. Whereas their participants highlight IDE-centered integration, our findings reveal a stronger need for CI/CD-level integration in non-technological companies.

Other work focuses on open-source settings. Beller et al. [32] show that only about half of popular OSS projects use static analysis tools and that configurations are rarely customized—consistent with our observation that many companies apply SAST with little contextual tailoring. Vassallo et al. [24] further demonstrate that developer engagement varies across contexts (IDE, CI, code review). Our results extend this view by showing that warnings must also adapt to different execution environments and organizational maturity levels.

Closely related is Seal et al. [11], who combine a survey and interviews to understand practitioners' perspectives on SAST tools. Unlike our organizational-level lens, their work highlights individual developers' concerns—particularly fear of false negatives and distrust in tool reports. Despite these differences, both studies converge on the need for clearer explanations, richer contextual information, and better workflow integration.

Positioning Our Study.: Our study contributes a complementary perspective by focusing on non-technological companies that develop enterprise systems to support their core business and by applying Socio-Technical Grounded Theory (STGT) to derive an emergent model of SAST adoption. In particular, we highlight (i) the organizational dynamics of overconfidence and reliance on infrastructure, (ii) the impact of legacy design decisions and cross-system dependencies on the feasibility of security remediation, and (iii) the need for contextualized, pipeline-aware SAST integration that respects both technical and organizational constraints. We recognize that some of our findings overlap with insights from prior studies—a result that is expected when using grounded theory [17]. What is most striking, however, is that concerns identified a decade ago (e.g., false positives and the need for clearer error messages) [6] still emerged in our data and in more recent papers [10], [11], [31].

VII. Conclusions

This paper reported the results of an empirical study investigating the adoption of SAST tools in non-technological companies. The findings complement existing literature by examining this phenomenon through the lens of organizations that build and maintain enterprise systems to support their core business. Using insights drawn from three focus groups—one in each participating company—we developed an emergent, grounded theoretical model that captures the socio-technical factors influencing SAST adoption. These hypotheses may assist organizations in adopting SAST tools more effectively, inform the designers of state-of-the-art analysis tools on how to improve usability and reporting quality, and inspire new research directions aimed at increasing the practical value of static analysis for detecting vulnerable code.

References

[1] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering - Design Principles and Practical Applications*. Wiley,

2010. [Online]. Available: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470474246.html

[2] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: why do java developers struggle with cryptography apis?" in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, L. K. Dillon, W. Visser, and L. A. Williams, Eds. ACM, 2016, pp. 935–946. [Online]. Available: https://doi.org/10.1145/2884781.2884790

[3] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, A. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 73–84. [Online]. Available: https://doi.org/10.1145/2508859.2516693

[4] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben, "Why eve and mallory love android: an analysis of android SSL (in)security," in *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 50–61. [Online]. Available: https://doi.org/10.1145/2382196.2382205

[5] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: validating SSL certificates in non-browser software," in *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 38–49. [Online]. Available: https://doi.org/10.1145/2382196.2382204

[6] B. Johnson, Y. Song, E. R. Murphy-Hill, and R. W. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, D. Notkin, B. H. C. Cheng, and K. Pohl, Eds. IEEE Computer Society, 2013, pp. 672–681. [Online]. Available: https://doi.org/10.1109/ICSE.2013.6606613

[7] J. Smith, B. Johnson, E. R. 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, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, E. D. Nitto, M. Harman, and P. Heymans, Eds. ACM, 2015, pp. 248–259. [Online]. Available: https://doi.org/10.1145/2786805.2786812

[8] T. Barik, Y. Song, B. Johnson, and E. R. Murphy-Hill, "From quick fixes to slow fixes: Reimagining static analysis resolutions to enable design space exploration," in *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*. IEEE Computer Society, 2016, pp. 211–221. [Online]. Available: https://doi.org/10.1109/ICSME.2016.63

[9] J. Smith, B. Johnson, E. R. Murphy-Hill, B. Chu, and H. R. Lipford, "How developers diagnose potential security vulnerabilities with a static analysis tool," *IEEE Trans. Software Eng.*, vol. 45, no. 9, pp. 877–897, 2019. [Online]. Available: https://doi.org/10.1109/TSE.2018.2810116

[10] J. Smith, L. N. Q. Do, and E. R. Murphy-Hill, "Why can't johnny fix vulnerabilities: A usability evaluation of static analysis tools for security," in *Sixteenth Symposium on Usable Privacy and Security, SOUPS 2020, August 7-11, 2020*, H. R. Lipford and S. Chiasson, Eds. USENIX Association, 2020, pp. 221–238. [Online]. Available: https://www.usenix.org/conference/soups2020/presentation/smith

[11] A. S. Ami, K. Moran, D. Poshyvanyk, and A. Nadkarni, ""false negative - that one is going to kill you": Understanding industry perspectives of static analysis based security testing," in *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*. IEEE, 2024, pp. 3979–3997. [Online]. Available: https://doi.org/10.1109/SP54263.2024.00019

[12] M. Christakis and C. Bird, "What developers want and need from program analysis: an empirical study," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, D. Lo, S. Apel, and S. Khurshid, Eds. ACM, 2016, pp. 332–343. [Online]. Available: https://doi.org/10.1145/2970276.2970347

[13] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from building static analysis tools at google," *Commun. ACM*, vol. 61, no. 4, pp. 58–66, 2018. [Online]. Available: https://doi.org/10.1145/3188720

[14] J. Kontio, J. Bragge, and L. Lehtola, "The focus group method as an empirical tool in software engineering," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, 2008, pp. 93–116. [Online]. Available: https://doi.org/10.1007/978-1-84800-044-5_4

[15] R. O'Connor, "Using grounded theory coding mechanisms to analyze case study and focus group data in the context of software process research," in *Research Methodologies, Innovations and Philosophies in Software Systems Engineering and Information Systems*, M. Mora *et al.*, Eds. IGI Global Scientific Publishing, 2012, pp. 256–270. [Online]. Available: https://doi.org/10.4018/978-1-4666-0179-6.ch013

[16] R. Hoda, "Socio-technical grounded theory for software engineering," *IEEE Trans. Software Eng.*, vol. 48, no. 10, pp. 3808–3832, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2021.3106280

[17] ——, *Qualitative Research with Socio-Technical Grounded Theory: A Practical Guide to Qualitative Data Analysis and Theory Development in the Digital World*. Springer, 2024.

[18] D. Hidellaarachchi, J. C. Grundy, R. Hoda, and I. Mueller, "The influence of human aspects on requirements engineering-related activities: Software practitioners' perspective," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, pp. 108:1–108:37, 2023. [Online]. Available: https://doi.org/10.1145/3546943

[19] K. Madampe, R. Hoda, and J. Grundy, "A framework for emotion-oriented requirements change handling in agile software engineering," *IEEE Trans. Software Eng.*, vol. 49, no. 5, pp. 3325–3343, 2023. [Online]. Available: https://doi.org/10.1109/TSE.2023.3253145

[20] B. G. Glaser and A. L. Strauss, *Discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017.

[21] A. Strauss and J. Corbin, *Basics of qualitative research*. Sage publications, 1990.

[22] K. Charmaz, *Constructing grounded theory: A practical guide through qualitative analysis*. sage, 2006.

[23] A. Habib and M. Pradel, "How many of all bugs do we find? a study of static bug detectors," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, M. Huchard, C. Kästner, and G. Fraser, Eds. ACM, 2018, pp. 317–328. [Online]. Available: https://doi.org/10.1145/3238147.3238213

[24] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, "How developers engage with static analysis tools in different contexts," *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1419–1457, 2020. [Online]. Available: https://doi.org/10.1007/s10664-019-09750-5

[25] Z. Guo, T. Tan, S. Liu, X. Liu, W. Lai, Y. Yang, Y. Li, L. Chen, W. Dong, and Y. Zhou, "Mitigating false positive static analysis warnings: Progress, challenges, and opportunities," *IEEE Trans. Software Eng.*, vol. 49, no. 12, pp. 5154–5188, 2023. [Online]. Available: https://doi.org/10.1109/TSE.2023.3329667

[26] L. Prates and R. Pereira, "Devsecops practices and tools," *International Journal of Information Security*, vol. 24, no. 1, pp. 1–25, 2025.

[27] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 121–136. [Online]. Available: https://doi.org/10.1109/SP.2017.31

[28] H. Zhang, S. Wang, H. Li, T. Chen, and A. E. Hassan, "A study of C/C++ code weaknesses on stack overflow," *IEEE Trans. Software Eng.*, vol. 48, no. 7, pp. 2359–2375, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2021.3058985

[29] S. Hamer, M. d'Amorim, and L. A. Williams, "Just another copy and paste? comparing the security vulnerabilities of chatgpt generated code and stackoverflow answers," in *IEEE Security and Privacy, SP 2024 - Workshops, San Francisco, CA, USA, May 23, 2024*. IEEE, 2024, pp. 87–94. [Online]. Available: https://doi.org/10.1109/SPW63631.2024.00014

[30] J. Jesus, L. Amaral, and R. Bonifácio, "Cryptographic api misuses in industry: A case study on prevalence and remediation," in *Anais Estendidos do XXV Simpósio Brasileiro de Cibersegurança*. Porto Alegre, RS, Brasil: SBC, 2025, pp. 405–413. [Online]. Available: https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/36784

[31] M. Nachtigall, M. Schlichtig, and E. Bodden, "A large-scale study of usability criteria addressed by static analysis tools," in *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*, S. Ryu and Y. Smaragdakis, Eds. ACM, 2022, pp. 532–543. [Online]. Available: https://doi.org/10.1145/3533767.3534374

[32] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, “Analyzing the state of static analysis: A large-scale evaluation in open source software,” in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*. IEEE Computer Society, 2016, pp. 470–481. [Online]. Available: https://doi.org/10.1109/SANER.2016.105