# Software security during modern code review: The developer's perspective

Braz, Larissa ; Bacchelli, Alberto

DOI: https://doi.org/10.1145/3540250.3549135

# Software Security during Modern Code Review: The Developer's Perspective

Larissa Braz
University of Zurich
Switzerland
larissa@ifi.uzh.ch

Alberto Bacchelli
University of Zurich
Switzerland
bacchelli@ifi.uzh.ch

## ABSTRACT

To avoid software vulnerabilities, organizations are shifting security to earlier stages of the software development, such as at code review time. In this paper, we aim to understand the developers' perspective on assessing software security during code review, the challenges they encounter, and the support that companies and projects provide. To this end, we conduct a two-step investigation: we interview 10 professional developers and survey 182 practitioners about software security assessment during code review. The outcome is an overview of how developers perceive software security during code review and a set of identified challenges. Our study revealed that most developers do not immediately report to focus on security issues during code review. Only after being asked about software security, developers state to always consider it during review and acknowledge its importance. Most companies do not provide security training, yet expect developers to still ensure security during reviews. Accordingly, developers report the lack of training and security knowledge as the main challenges they face when checking for security issues. In addition, they have challenges with third-party libraries and to identify interactions between parts of code that could have security implications. Moreover, security may be disregarded during reviews due to developers' assumptions about the security dynamic of the application they develop.
**Data and materials:** https://doi.org/10.5281/zenodo.6875435

## CCS CONCEPTS

• **Security and privacy → Software security engineering**.

## KEYWORDS

code review, security, software vulnerabilities

**ACM Reference Format:**
Larissa Braz and Alberto Bacchelli. 2022. Software Security during Modern Code Review: The Developer's Perspective. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3549135

## 1 INTRODUCTION

A software vulnerability is a security flaw, glitch, or weakness found in software code that could be exploited by an attacker [23] to cause harm to the stakeholders of a software system [28]. To avoid vulnerabilities in software systems, organizations are shifting security "left," that is, to earlier stages of software development, such as during code review [1]. Code review is a widely agreed-on practice [12] recognized as a valuable tool for reducing software defects and improving the quality of software projects [2, 3, 8]. Previous studies show that code review is also an important practice for detecting and fixing security bugs earlier [41, 67] and has positive effects on Secure Development of Applications (*SDA*) [42, 43, 57]. An application that has not been reviewed for security gaps is likely to have problems virtually at 100% [51].

Previous studies [49, 73, 76, 77] have reported and investigated the limited adoption of *SDA* and investigated the developer's perspective on security. However, as software security seems to increasingly gain more space among developers' responsibilities, further studies are needed to investigate this left-shift. In particular, in this paper, we focus on the current state of security during code review according to practitioners. Our definition of security is rooted in the definition of software vulnerabilities. In fact, ensuring security during code review means finding and proposing fixes for security vulnerabilities in the code under review to avoid turning the application insecure [51]. Moreover, findings on this topic can provide insight for practitioners, organizations, and researchers. Developers and other software project stakeholders can use our insights to improve their code review practices to better ensure security during code review. Organizations can use evidence about developers' challenges during code reviews to update and improve their approaches and methodologies toward achieving and maintaining secure applications. Finally, researchers can focus their attention on developers' challenges to facilitate secure code review.

We set up our investigation as an explorative study. Thus, instead of starting with preset hypotheses on how software security is or should be addressed during code reviews, we investigate: (1) the developer's perspective on assessing software security during code review; (2) the expectations and support that companies and projects have on this process; and (3) what problems software engineers face when evaluating software security during reviews.

We structure our investigation into two steps. In the first step, we interview ten professional developers by means of 30 minutes semi-structured interviews. In the second step, we design and disseminate an online survey to collect developers' perceptions, practices, and challenges concerning software security inspection during modern code review. The survey received a total of 182 valid answers. Among the survey respondents, 78% (142) report being software

*ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore*

developers currently, 84% (153) have three years or more of professional development experience, and 62% (113) conduct code reviews at least several times per week.

Our study reveals that most participants (105) develop security-sensitive software systems; yet, when asked which issues they focus on during code review, only 9 survey respondents explicitly mention security. Only after we mention security, 111 respondents state to *always* consider it during code review. This stark mismatch may indicate that security is not a priority during review and may be assessed less frequently than reported. Moreover, developers may disregard security aspects during reviews due to their assumptions about the security dynamic of the application they develop.

Concerning companies, most respondents (126) work for companies or projects that expect developers to ensure security, including through code review. However, the vast majority of the respondents (149) think that companies should do more to support secure practices. For example, 122 respondents are not acknowledged for performing secure code reviews. Moreover, two-thirds of the respondents state that companies do not provide security training, rather just allow developers to acquire security skills by themselves during working hours (reported by 95 respondents).

Indeed, when it comes to challenges of ensuring security in code review, respondents mostly report lack of training and knowledge as the main issue they face (mentioned by 44 respondents). Second, the use of third-party libraries is problematic because they are not always present in the code developers maintain; thus, checking them actively during code review is not possible (mentioned by eight respondents). Finally, some respondents explained that code review is not the ideal phase of the development process to detect vulnerabilities. For instance, some vulnerabilities require the execution of the code to appear. Moreover, developers find it hard to identify interactions among parts of code that could have security implications at low-level inspections.

Based on the findings of our study, we propose a series of recommendations and areas for future research.

## 2 BACKGROUND AND RELATED WORK

In this section, we review the Software Engineering (SE) research literature on topics investigating software security and the developer-related factors that influence vulnerability detection.

**Security in the Development Lifecycle.** In 2018, the McAfee institution [40] reported that, daily, 80 billion malicious scans look for vulnerable targets, and 780 thousand records are lost to hacking. However, security experts still have to motivate and convince developers of the importance of finding vulnerabilities [66]. Later, GitLab performed a survey with over 3,650 respondents from 21 countries [1]. In their survey, security experts reported that it is a software developer's job to develop secure code, but only less than half of the developers can actually detect vulnerabilities.

Security often fails because users either misunderstand the security implications of their actions or turn off security features to workaround usability problems [10]. In fact, security usability issues also affect developers. For instance, storing user login data and authenticating users is prone to security issues due to the high complexity of the technologies and concepts involved in the process [33]. Usability issues existing in such security APIs force non-security expert programmers to misuse these APIs and introduce security vulnerabilities to applications they develop [71]. Developer-Centred Security (*DCS*) studies have addressed some of the developers' needs and attempted to apply existing Human-Computer Interaction methodologies and to adopt well-established usable security measures to software development [33, 47, 75]. However, Tahaei and Vaniea [64] report a lack of research on several aspects of *DCS*, including how to make security a business value and security often being ignored because it is a secondary requirement.

In 2004, Microsoft released a Trustworthy Computing Security Development Lifecycle initiative [74], a process adopted by the company to develop software that needs to withstand malicious attacks. The process included the addition of a series of security-focused activities to each phase of Microsoft's software development process. Lipner [38] investigated the effectiveness of Microsoft's initiative and found that the practice of Secure Development of Applications (*SDA*) provides security benefits.

In 2007, Woon and Kankanhalli [73] conducted a field survey of 184 information system professionals to investigate the factors that may influence the intention of the participants to practice Secure Development of Applications (*SDA*), *i.e.,* incorporate security as part of the application development lifecycle. Their results show evidence that participants' intention was determined primarily by attitude, followed by product usefulness and subjective norm. Moreover, self-efficacy and facilitating conditions did not appear to impact the information system professionals' intention to practice *SDA*. In addition, participants reported that companies did not facilitate *SDA* practice other than allowing the respondents to attend seminars on the topic. Professionals could decide whether to practice it, but the organizations did not give them rewards or recognition for their extra efforts. Moreover, Woon and Kankanhalli [73] reported limited adoption of *SDA* and a lack of studies exploring the phenomenon. Later, Xie et al. [77] identified an "it is not my responsibility" attitude from the developers towards *SDA*. Several studies [5, 49, 76] found that developers prioritize more-visible functional requirements or even easy-to-measure activities, *e.g.,* closing bug tracking tickets, over security. On the contrary, Christakis and Bird [17] reported that developers care more about security than other reliability issues.

Smith et al. [60] advocate that static analysis tools detect vulnerabilities and help developers resolve them. Previous studies have proposed and improved tooling support according to developers' needs [6, 7, 59], but tools are still generally poorly adopted by developers [64] as they are confusing for developers to use [60].

The aforementioned studies [5, 17, 49, 73, 76, 77] provide insights on *SDA* and how developers perceive security. However, they might not represent the current development scenario. For instance, Woon and Kankanhalli [73] study was conducted 15 years ago, and [77] was performed 11 years ago. Nowadays, as software security seems to be gaining more space among developers' responsibilities, in this paper, we investigate whether their findings still hold. To this aim, we investigate the current developer's perspective on this phenomenon and whether they assess software security during code reviews. We created our initial interviews guideline according to the findings of these studies.

**Code Review and Security.** Code review is a way to manually inspect source code by developers other than the author [19]. In its contemporary practice, code review is asynchronous, tool- and change-based [11], and widely used across companies [8, 56] as well as community-driven projects [54, 55]. Bacchelli and Bird [8] surveyed developers and managers' expectations regarding code review at Microsoft. Among the observations they collected during interviews, a senior developer stated: "I've seen quite a few code reviews where someone commented on formatting while missing the fact that there were security issues."

Edmundson et al. [24] stated that manual code review could be expensive and impractical due to the need for several reviewers to inspect a piece of code to find a vulnerability. On the contrary, Weir et al. [69] interviewed twelve industry experts to investigate how to improve the security skills of mobile app developers. Some of the techniques recommended included code reviews. In this vein, the OWASP institution [51] defined secure code review as probably the single-most effective technique for identifying security bugs early in the system development lifecycle.

Braz et al. [16] investigated to what extent software developers can detect vulnerabilities during code reviews. They found that several developers often miss a popular and easy-to-detect vulnerability when reviewing code; yet, when explicitly informed about the presence of a vulnerability in the change, a significant portion of the additional developers could identify it. Later, Braz et al. [14] investigated whether and to what extent instructing developers to focus on security issues and providing security checklists during code reviews can support the detection of software vulnerabilities. They found that developers' mental attitude plays a role in detecting software vulnerabilities during code reviews. The effect of security instructions provided evidence that vulnerability detection could be triggered with proper security considerations, such as security standards for code reviews.

These studies [14, 16] have investigated the developer's ability to detect vulnerabilities during a practical code review task. They also provided the first evidence on how developers could perceive security assessment during code review. In this work, we continue on this research path, focusing specifically on developers' perceptions as a way to generate knowledge to inform current practices and research.

# 3 METHODOLOGY

Overall, our research aims to understand the developers' perspective on assessing software security during code review and the support companies and projects provide to this process. We base our study on semi-structured interviews and a survey we devised and disseminated to collect different types of self-reported data.

## 3.1 Research Questions

Our investigation is structured around three research questions. We incorporated them in the interview guidelines and the survey.

As software security seems to be gaining more space among developers' responsibilities, we first investigate the developer's perspective on this phenomenon and whether the developers in fact assess software security during code reviews:

**RQ$_1$.** *What is the current developer's perspective on ensuring software security during code review?*

Organizations are shifting security to earlier stages of software development, increasing developers' responsibility around security. We investigate the support that companies and projects provide to the software security assessment during code reviews:

**RQ$_2$.** *To what extent do companies/projects support security assessment during code review?*

Finally, we focus on the challenges developers face when ensuring software security during code reviews. Such an understanding helps us obtain a catalog of reasons why this practice is not adopted and allow us to propose recommendations to mitigate them.

**RQ$_3$.** *What are the main challenges experienced when ensuring security during modern code reviews?*

## 3.2 Semi-structured Interview Design

We performed the first part of our study as a set of one-to-one semi-structured interviews [37] with professional software developers of different backgrounds. Semi-structured interviews use of an interview guide that contains general groupings of topics and questions rather than a pre-determined set and order of questions [8, 37]. They are often used in an exploratory context to "find out what is happening [and] to seek new insights" [70]. The guideline was iteratively refined after each interview, particularly when developers started providing answers very similar to the earlier ones, thus reaching a saturation effect. The main path of the final interview guideline has 28 questions. Table 1 shows two of the questions we asked participants during the interviews. Following, we describe the guideline's main points.

**(1) Introduction and background questions.** In the introduction, we spend a few minutes explaining who we are and our research — *without mentioning security*. We review the main points of the consent form they signed prior to the interviews and answer any questions they might have. For instance, we remind them they were allowed to stop the interview at any moment and also skip any question they did not feel like answering. We also explain that there were no right or wrong answers. Following, we start recording the interview and ask the participants questions about their background, such as their current role and how many years of experience in programming they have. The discussion of participant background serves as an icebreaker and also provides some context for later in the interview.

**Table 1: Sample of the interviews questions.**

| | |
|---|---|
| Q15 | To what extent do you consider security aspects when reviewing code? |
| Q17 | Do you think ensuring security during code reviews is challenging? If yes, why do you think it is challenging? |

**(2) Code review questions.** In this part of the interview, we ask general questions about code reviews. First, we ask participants if they perform code reviews as part of their current job or whether they have previous experience with it. Although we explicitly state the interview is about code reviews when inviting participants, it could happen that a participant answer they have no experience with this activity whatsoever. In this case, we move the interview to the wrap-up bulk. Participants that confirm they have conducted code review sessions continue to answer questions about their experiences. For instance, we ask the participant why they perform code reviews and what type of issues they look for when reviewing. At this moment, we observe whether participants mention security or vulnerabilities by themselves, *i.e.,* whether security is spontaneously mentioned without being primed.

**(3) Security questions.** We ask participants if they are familiar with the term "software vulnerabilities" and their definition of it. For participants unfamiliar with the term, we read OWASP's [28] definition and ask if they agree with it. The subsequent questions and discussion regarding security are in this context of vulnerability detection. We ask participants whether they consider security when reviewing code. To those who confirm they do it, we ask security-specific questions about their practices and experiences. To participants that do not consider security during reviews, we ask questions about security in other activities of the software development process. We then ask participants about the challenges to ensure security in their reviews and other development activities.

**(4) Wrap-up.** We conclude the interviews with some basic questions, for instance: "Is there anything you want to tell us about code reviews and security that we did not cover before?" and "Is there anything you want to ask us?". Finally, we thank them.

We conducted the interviews through a video-conferencing application. With consent, we recorded the interviews, assuring the participants of anonymity. The audio of each interview was then transcribed and broken up into smaller coherent units for subsequent analysis. The transcripts of the interviews are available [15].

## 3.3 Survey Design

In the second part of our study, we conducted a survey with developers who have experience with code reviews. Our goal was to validate and expand on the findings collected through the semi-structured interviews, and further answer our research questions (see section 3.1). We built and ran the survey on Qualtrics [48].

The survey had 22 questions compulsory questions, organized into four open-ended questions, three multiple-choice, 15 single-choice, and three 5-point Likert scale grids. These grids asked participants to rate 8, 7, and 4 items on a 5-point Likert scale (ranging from 'strongly disagree' to 'strongly agree'). The multiple-choice and rate questions had randomized answer options and statements. Table 2 shows an example of a closed and an open-ended question. In the following, we detail the survey's design and how the participant flows through each block of questions. Each block corresponds to at least one different page, and returning to previous pages is not allowed. The complete survey is available in the accompanying material [15].

**(1) Welcome page.** On the first page of the survey, we provide participants with information about the study. We do not inform the participants about the study's final focus on software vulnerabilities to avoid they subconsciously changing their answers to fit it (*i.e.,* demand characteristics [45]). We inform participants about the data handling policy, ask for their consent to use their data, and inform them that they are allowed to drop out at any time.

**(2) Qualification questions.** We ask participants two random technical questions to screen non-programmers out of our study. We selected three technical questions from Danilova et al. [21]'s list as they offer only minimal overhead for the participants with actual programming skills.

**(3) Code review experience and practices.** In this block, we ask questions to gather information about their code reviews' experiences and practices, including the reasons why they perform the reviews and the type of issues they have reported in their last code review sessions. At this point, *we do not ask security-specific questions.* This way, we can identify participants who spontaneously report to include security as part of their code reviews.

**(3) Security knowledge and practices during code reviews.** In this step, we ask participants whether they are familiar with the term *software vulnerabilities* and provide them with its definition [28]. The subsequent questions regarding security are in this context of vulnerability detection. We ask questions to gather information about factors that may affect how the participants address security during code reviews, such as their security knowledge, practices, and company and team culture. Most of the questions are closed in a Likert scale format. Depending on a participant's answers, some questions were filtered and not presented to avoid asking unnecessary questions. Different from the interview, in the survey we did not explicitly ask participants about the security process followed within their companies. We excluded this question to avoid overloading survey respondents and to increase response quality [30].

**(4) Demographics.** We ask participants questions to collect demographic information and confounding factors, such as the highest obtained education and years of professional experience (all questions are available in the replication package [15]). This information is mandatory to fill in as collecting it helps us identify which portion of the developer population is represented by our respondents [26].

**(5) Feedback and closing.** We ask for the participant's feedback on the survey. We also ask if they would like to share their anonymous data in a public research dataset and receive the study results.

## 3.4 Interviews and Open Answers Analysis

We performed two open card sortings [63] to extract emerging themes from the interviews and open answers of the survey. This allowed organizing the codes into hierarchies to deduce a higher level of data abstraction. We followed the same process in both

**Table 2: Sample of the survey questions.**

| | |
|---|---|
| Q6 | What types of issues do you focus on during your code review sessions? |
| Q12.4 | It is part of the developer's job to ensure the security of the application *(strongly disagree, disagree, neither agree nor disagree, agree, strongly agree, not applicable*). |

card sortings: the first authors and an external researcher created self-contained units, then sorted them into themes. To ensure the themes' integrity, the authors interactively sorted the units several times. After review by the last author and discussions, we reached the final themes. Through the discussions, we evaluated controversial answers, reduced potential bias caused by wrong interpretations of a participant's answer, and strengthened the confidence in the card sort output (available in our replication package [15]).

### 3.5 Recruiting Participants

To recruit participants for the interviews, we contact our professional networks as well as ask for participation on practitioners' web forums. We disseminated the online survey out through practitioners' web forums, IRC communication channels, direct authors' contacts from their professional networks, as well as their social media accounts (*e.g.,* Twitter, Facebook). We did not reveal the security angle of the study; instead, we explained that the study was about expectations and practices in code review. We also introduced a donation-based incentive of 20 USD and 2 USD to a charity per participant of the interviews and the survey, respectively.

## 4 RESULTS

In this section, we report the results of our investigation.

### 4.1 Participants

In total, we interviewed ten software developers with up to 13 years of industry experience. Among the interviewees, three also have a technical lead position within their team. All interviewees work in different companies of distinct dimensions: three startups, three mid-sized software engineer companies, and four large software engineer companies. One interviewee (I1) is also active in the open source community. Two interviewees are located in Brazil, one in Canada, one in England, one in Portugal, one in the Netherlands, and four in Switzerland. All interviewees perform code reviews at least three times a week. Table 3 summarizes the background of the interviewees.

A total of 329 people started our online survey through the provided link. Of these, 121 did not complete it; thus, we removed their entries from the results dataset. A total of 208 people completed all survey blocks. We removed 24 respondents who reported not performing code reviews and manually inspected five respondents who incorrectly answered at least one qualification question, removing other two respondents. After applying these exclusion criteria, data from 182 respondents could be used for the analyses.

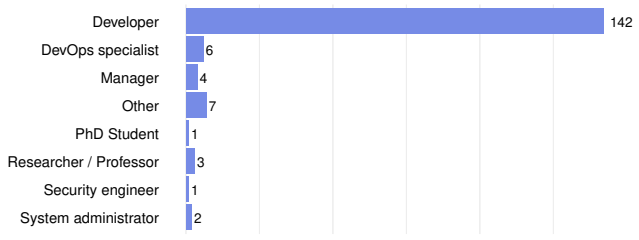


**Figure 1: Job distribution among employed respondents.**

**Table 3: Profiles of the interviewed professional developers.**

| ID | Domain | Company/ Project | Experience (in years) | Security Expert |
|---|---|---|---|---|
| *I1* | Indust./OSS | Mid SW Comp. | 9 | ✓ |
| *I2* | Industry | Startup | 1 | |
| *I3* | Industry | Large SW Comp. | 1 | |
| *I4* | Industry | Large SW Comp. | 11 | |
| *I5* | Industry | Startup | 1 | |
| *I6* | Industry | Large SW Comp. | 13 | |
| *I7* | Industry | Mid SW Comp. | 6 | |
| *I8* | Industry | Mid SW Comp. | 7 | |
| *I9* | Industry | Startup | 2 | |
| *I10* | Industry | Large SW Comp. | 8 | |

Figure 1 shows the current positions of respondents with an employment and Figure 2 presents the respondents' experience and practice. In total, 178 respondents reported being employed. Most respondents currently have a developer role (78%), distributed as: front-end (2 respondents), back-end (45), full-stack (78), mobile (2), embedded applications or devices (5), and others (10). Moreover, most respondents (84%) report more than two years of professional development experience, and to program (91%), to design (68%), and to review code (62%) at least several times per week.

### 4.2 $RQ_1$. Security during Code Reviews

Our first research question seeks to investigate the developer's perspective on software security and whether the developers in fact assess software security during code reviews.

**Security focus during code review.** When asked—as an open question— what they focus on during code review, only 9 out of the 182 survey respondents explicitly mentioned security. Instead, other non-functional qualities of the code were mentioned more frequently (*e.g.,* 23 respondents reported performance as an issue they focus on during code review). However, after being prompted about vulnerabilities, 111 respondents reported to *always* consider security during reviews. Most survey respondents (81%) reported they can decide what aspects/issues to inspect during reviews (*i.e.,* in the end, ensuring security during reviews is their choice), and 81% of them state it is their responsibility to look for vulnerabilities during reviews.

During the interviews, two interviewees mentioned, without being prompted, to focus on security. I4 explained: "There are collateral effects that can be undesirable for the change…like performance, security, maintainability, testability, extensibility. All these things that can be tested and generally are tested." Eight interviewees reported to be familiar with the term software vulnerability and provided their own definition, which was in line with ours in all cases (see Section 3.2). This was also the case of I1, who reported to have a security background. The two remaining interviewees initially reported no familiarity with the term but agreed with the definition we provided. One of them further explained that they were initially unsure about the correct definition of the term, but our definition confirmed their initial opinion.

After being prompted about software vulnerabilities, five more interviewees highlighted the importance of considering security

during code reviews. Out of these five interviewees, I7 stated: "I consider it a lot, because I already worked for companies that we had problems related to it: the security of the data, big companies." Survey participants and interviewees used various terms in their answers refer to the application's security. This was the case of the previous answer given by I7 where they mentioned "security of the data" when asked whether they consider security during code reviews, a follow-up question to the discussion on the software vulnerability definition. In fact, vulnerabilities in the code may lead to insecure data; for instance, a Use of a Broken or Risky Cryptographic Algorithm (CWE 327) [50] flaw in the code can lead to the exploit of sensitive data.

Moreover, two interviewees reported using automatic tools to detect software vulnerabilities, not only during code review but overall during development activities. For instance, I2 said: "this is automatically checked. There is a tool [that] is part of the automatic check, if that change that I am introducing, if it is exposing any credential, if there is any hardcoded password, if I let any security permission leak."

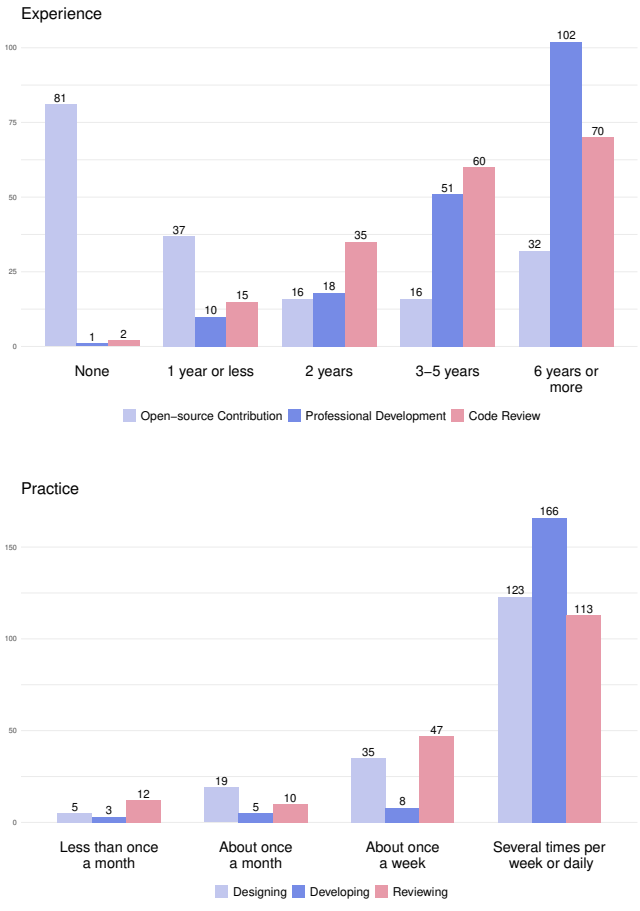


**Figure 2: respondents' demographics (absolute numbers).**

**Participants' familiarity with security.** The familiarity of our study's participants with security and its concepts plays an important role in our results' interpretation. Among the survey respondents, 37 stated to be unfamiliar or unsure about the term software vulnerability, while 145 reported being familiar with it.

Two interviewees reported to not be familiar with the term software vulnerability. Yet, they demonstrated to be security-aware after listening to the software vulnerability definition. Except for I1 (who reported to have worked with security), the interviewees pointed up not being security experts. I5 explained: "[security] is a topic that most of engineers tend to not look too much into it."

> ***Observation 1.*** *Developers are familiar with the term software vulnerability and acknowledge the importance of ensuring security during code reviews. Yet, the vast majority of participants mentioned security as a key focus during code review only after we explicitly asked them about this topic.*

**Developers' assumptions about application's security.** The interview format allowed us to reveal potentially dangerous assumptions developers make. In total, seven interviewees assumed that security is the responsibility of another software component or team. For instance, I1 explained: "I do not worry much [about security] … I am supposing that another part of the system already dealt with it, like already dealt with the security and, at this point, I am relatively secure and should not worry much about it." I9 stated: "I am more a front-end developer, we deal less with security matters. I think [considering security during code reviews] should be more for the back end, right? They deal more with data." In addition, I6 said "I am a back-end engineer … I am not a front-end person, I do not have front-end expertise, so verifying accessibility [vulnerabilities], for example, is something I could not easily do."

Two interviewees revealed that because they develop internal code, they do not extensively consider security during code review. For instance, I3 said: "I think about it for like 3 seconds and I see that there is nothing to worry about. …again the code I wrote most, like 95% of the time, is internal code that will never go outside."

In contrast to the interviewees, most survey participants reported that ensuring security is part of their jobs. In total, 126 (81%) survey respondents agree with the statement:"It is my responsibility to look for vulnerabilities during code reviews." In addition, 166 agree with the statement: "It is part of the developer's job to ensure the security of the application." Yet, the difference between interviewees and survey respondents may also be explained by having prompted the latter about vulnerabilities (see section 3.3).

> ***Observation 2.*** *Interviewees stated to disregard security aspects during code reviews due to their assumptions about the security dynamic of the application they develop.*

## 4.3 RQ$_2$. Support to Ensure Security on Reviews

Our second research question seeks to investigate the support that companies and projects provide to the software security assessment during code reviews.

Most survey respondents (92%) reported to review code because they see value in code review, while only 15 of the respondents reported they conduct this activity because they are told to do so (*i.e.,* due to company/team policy). Figure 3 presents the survey respondents' scaled rates about their experiences, practices, and company/projects culture. Overall, most survey respondents (58%) reported that the main application they develop for is security-sensitive; yet, fewer (45%) respondents reported their companies and projects to consider security as a main concern. In total, 85 (47%) respondents reported their company/project to have a dedicated security team, and only 12% (21) of the respondents have their changes reviewed by security experts.

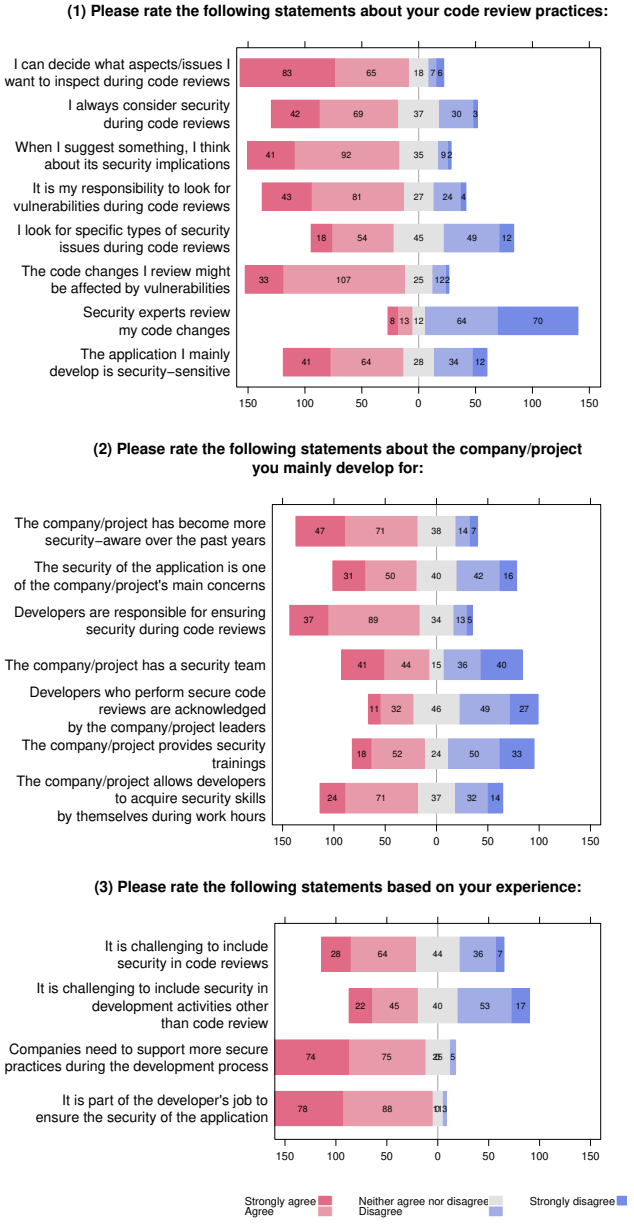


**Figure 3: Participant's scaled rates about their experiences, practices and company/projects culture.**

In total, 147 (81%) respondents reported that the code changes they review might be affected by software vulnerabilities. However, only 12% of the code changes are reviewed by security experts. For 69% (126) of the respondents, developers are responsible for ensuring security during code reviews in their company/project. However, only 24% (43) of the respondents feel like the companies and project leaders acknowledge developers that perform secure code reviews.

Furthermore, 38% (70) of the survey respondents reported that their company/project provides security training, but 52% (95) of the companies/projects allow developers to acquire security skills by themselves during work hours. For 82% of the respondents, companies need to support more secure practices during the development process, including during code review.

Two interviewees commented about how their companies became more security-aware over time. For instance, I7 told us about how their company started as a small business and how its growth impacted the security of the application: "When I joined, the application supported very few devices … with very low sales, so it was not such a big concern, but [since we had an equipment] used by over 60,000 people … we started to have a certain kind of care. We [added] more security in the application."

Three interviewees reported that their companies have dedicated security teams. However, security reviews of every change might not be viable: "Unfortunately, I do not think we have enough people that are really expert in security to be able to do a scrutiny security verification in all code changes we have. …It is more scalable that you give a prototype to a security team that will do an analysis to see if any attack pattern is there" (I6).

> ***Observation 3***. *Respondents are expected to ensure security during code reviews. However, they are not acknowledged by companies/project leaders when they do it.*

## 4.4 RQ$_3$. Challenges to Ensure Security

Our third research question seeks to understand the challenges developers face when ensuring software security during code reviews. In addition, we also explore developers' recommendations on how to mitigate these challenges.

When asked whether it is challenging to ensure security during code review, seven interviewees agreed. More than that, "it can be hard even for experts to ensure security during reviews" (I7). In the survey, half (51%) of the respondents reported finding it challenging to ensure security at review time, while fewer 37% (67) reported it as challenging to do it in other development activities.

Interviewees reported lack of knowledge, the large amount of possible vulnerabilities and attack patterns, time constraints, and

human factors as their main challenges. I6 said "Because there are many, many ways of attacking, a lot more that a person who is not a specialist in the field will be able to check (...) the variability of security problems is too extensive. Because of that, it would take too much time for a nonspecialist person to verify it, besides verifying what the change needs to actually do." We asked survey respondents to tell us the challenges they face concerning security during code reviews as an open-text question. Respondents reported similar reasons as to the interviewees. Survey respondents mentioned lack of knowledge and time as the main reasons they find it challenging to ensure security during code reviews. For instance, the set size of possible vulnerabilities was also mentioned in the survey. R172 explained: "It is difficult to consider all of the possible security vulnerabilities, especially in larger ecosystems where a library may have a vulnerability that is not visible on the top layer." Figure 4 shows the challenges survey respondents reported to face concerning security during code reviews.

**No struggle to ensure security.** In total, four interviewees denied struggling to ensure software security during code reviews. For instance, I2 said: "I do not think [I struggle] because this is something that already existed, it is already part of the company." In addition, I6 said "I never felt like struggling. I think what already happened, that sometimes happens, is that because I am not a specialist, I end up having doubts about the security of something that I am seeing and I suggest that change goes through a review a bit more specific with a security team." On the other hand, three interviewees said to struggle in this activity. For example, I7 said: "Yes, a lot, because, although I know some points where I know the security issue is critical, I do not know everything. So, sometimes what I said before ( [small things]) go by unnoticed because sometimes I do not even know that is a point that can be exploited."

Two of the interviewees also reported to struggle to include security in other activities of the development lifecycle: "Security is something that we try to have in mind, but sometimes, I do not know, sometimes you need to deliver stuff for yesterday. Sometimes there is something in the middle of your way. Sometimes there is a certain hurry that [security] ends up unnoticed" (I7).


**Figure 4: Challenges developers face concerning security during code reviews.**

> ***Observation 4.*** *Developers deny struggling to ensure security during code reviews. Moreover, doing it depends on the change's size, complexity and context.*

**Insufficient security knowledge and training.** In our survey, the most frequently mentioned challenge (mentioned by 44 respondents) is insufficient security knowledge and training. For instance, R35 said: "lack of knowledge about the current vulnerabilities on the language/ framework/libraries of choice," and R86 explained that "developers often lack the low-level and/or platform expertise necessary to detect some vulnerabilities." R15 explained: "[security assessment] needs specialized knowledge (exploit techniques) and it can be hard to convince colleagues that there is a security [vulnerabilities] in their code without doing a demo if they do not have the necessary knowledge." Survey respondents also mentioned that vulnerabilities might not be easy to identify during the review: "Vulnerabilities are not always obvious just from looking at code, [they] sometimes slip through the cracks even when best practices are followed" (R86).

Respondents were also skeptical about developers' abilities to learn and perform secure code reviews. For instance, R154 explained "Dedicated security experts are not economically viable for most smaller companies / teams / project. Some training might help but it is probably an unrealistic for all developers to become competent enough in IT security. Addressing this issue requires better tool support (static code analysis) and improved frameworks / libraries / API, so that developers have less 'opportunity' to mess up."

**Vulnerabilities are hard to detect and keep up with.** The nature of software vulnerabilities also impacts its detection. In total, nine survey participants reported that vulnerabilities are subtle and difficult to detect, even when the best practices are followed. For instance, R86 explained that "vulnerabilities are not always obvious just from looking at code," and R154 said that "some vulnerabilities are very subtle (yet dangerous) and therefore easy to overlook. The code actually does not look 'bad'." In addition to that, the large amount of vulnerabilities and the constant discovery of new ones turns keeping up with security a hard task. For example, R31 reported the "ever-changing development landscape, new vulnerabilities are found in third-party packages every day" as a challenging factor to ensure security during review.

**Lack of codebase knowledge.** Lack of codebase knowledge was mentioned nine times. Survey respondents reported lack of knowledge about applications' architecture, domain, and components interaction as factors that influence the detection of vulnerabilities during reviews. For instance, R39 explained "I am not a security expert; my reviews will never be airtight on a security level. The level of threat is hard to determine since company-wide network policies and other teams' applications are also part of the ecosystem with details that are not necessarily known." In addition, R42 said "lack of knowledge of the overall domain (I work in a heavily regulated financial environment and I am relatively junior) with many existing security measures. Awareness of the interactions between these systems is desirable when inspecting security."

**Use of third-party libraries.** Lack of knowledge also applies to third-party libraries used in the application. The use of third-party libraries was mentioned by eight survey respondents as a challenge they face when ensuring security during code reviews. R86 said "Vulnerabilities are not always present in the code we maintain, often they are library/platform/integration issues."

**Security awareness at review time.** The results of $\mathbf{RQ}_1$ show that developers are familiar with security but do not focus on it during code reviews. Some developers acknowledge this scenario. In total, eight survey respondents mentioned the lack of security awareness during code reviews as a challenging factor to detect vulnerabilities during the review. For instance, R16 reported "general lack of security awareness from most developers" as a challenge they face. Moreover, R147 said that "[security] is typically not on the forefront of the mind of the author."

**Time and security interest.** Lack of time (six respondents), interest in security (one), and resources (two) were also mentioned as challenge factors developers face to ensure security at code review time. R62 said "[considering security during code reviews] is too time-consuming." In addition, R132 stated: "Lack of time/interest/funding for security-centric changes leads to security being an afterthought." Finally, R80 wrote: "So many changes happen that it can be hard to see how a small change affects the security. Security has not been thought of at the start and was hacked in."

**Human factors.** Survey participants also mentioned human factors, such as fatigue and focus, as factors that impact the detection of security issues during code reviews: "It is time pressure, fatigue, focus, and all these things, because if you have to review lots of codes, you will end up lowering your attention. If you are constantly interrupted during a code review, when you need to assemble the system in your head, assemble the change in your head along with the system and seeing how it works, that can get in the way" (I4).

**Code review limitations.** In total, five survey respondents also mentioned some limitations that code review poses to ensuring security. For instance, R64 mentioned "too much code to review" as a challenging factor to ensure security during reviews, and R112 said: " [I am] not always sure what to look for, nor sure if it is my responsibility instead of our security team's responsibility. Code reviews also show only a limited view of an application, and proper understanding of security flaws may require seeing how code interacts across the whole application." R86 wrote: "spotting vulnerabilities often requires deep understanding of the subject code which is not really required of code reviewers."

***Observation 5**. Insufficient security knowledge is the mainly reported challenge in ensuring security during code reviews. Knowledge regarding the application and its components, third-party libraries, human factors, and code review limitations also hinder security assessment at review time.*

# 5 LIMITATIONS

Assessing the validity of explorative qualitative research is challenging [32, 46]. Despite our best efforts, some limitations exist; in the following, we explain how we tried to minimize them.

To mitigate the possible limitation from missing control over subjects, we included questions to characterize our sample, such as experience and role (Block 4 in Section 3.3). We also removed participants who did not complete the experiment and reported not performing code reviews. We manually analyzed participants that incorrectly answered the qualification questions.

During the interviews, we might have led interviewees to provide more desirable answers [34]. To mitigate this issue, we challenged and triangulated our findings with the survey results. Our study participants might have given socially acceptable answers to appear in a positive light. To mitigate this social desirability bias [29], we informed participants that the responses would be anonymous and evaluated in a statistical form. To mitigate question-order effect [58] that might have led the survey respondents to specific answers, we randomized the order of answers of the multiple-choice questions and the statements of the scale rate questions.

We collected qualitative data from the interviews and the survey to understand developers' main challenges to ensure security during code reviews. We used the data to get insights into their perceived challenges and what they would do/recommend to do to mitigate such issues. We used different measurement techniques to mitigate *mono-method bias* [20]: we obtained qualitative results by employing card sorting on participants' responses to the survey's open-text questions (Block 3 in Section 3.3). We also used this technique on the interviews' transcripts.

We only collected data through interviews and surveys, which may not provide the full picture of developers' perceptions. To mitigate this limitation, one can examine code, design documents, issue tracking system contents, and other repositories. We hope that future research can be inspired by our results and triangulate selected findings with other data sources.

To obtain a diverse sample of participants, we invited software developers from several countries, organizations, education levels, experiences, and backgrounds. Even though our sample comprises several types of software developers and we found a large agreement concerning their perceptions, we cannot claim it is representative of all developers. If the study is repeated using different participants, the results may be different.

Participants could freely decide whether to participate in the study or not (self-selection). They were informed about the survey's topic (*code review*), an estimated duration for the participation, and offered a donation to a charity institution to encourage their participation. This could have biased the selection of participants as only participants who could spare enough time or were interested in the incentive might have participated. We tried to mitigate this risk by advertising through various channels.

# 6 DISCUSSION

We discuss how the findings on the developer's perception of software security code reviews can be used to better support the secure development of applications, and we outline opportunities for future work.

**A change in developer's perspective on security.** In the past, developers have shown an "it is not my responsibility" attitude towards software security [77] and limited adoption of Secure Development of Applications (*SDA*) [73]. However, software security seems to be gaining more space among developers' responsibilities. In fact, a study [17] reported that developers care more about security than other reliability issues. Following this trend, our results suggest a change in developer's attitude—they state it is their responsibility to ensure security during code reviews.

**Security needs to be better motivated.** Even though developers seem to have changed their attitude towards security, recent research [14, 16] has also provided initial evidence that security is not in the developers' mind "by default" when reviewing code. Our results corroborate this evidence: We found that developers do not have security in mind when describing their code review practices. A reason for this might be that developers lack motivation to be concerned about security during reviews. Indeed, companies and projects expect developers to ensure the security of their applications and allow them to acquire security knowledge during work hours, but they do not provide the means for that, *e.g.,* security training and workshops. Moreover, leaders do not acknowledge developers that do secure code reviews.

This situation raises questions on the effectiveness of how companies motivate developers in the software development process, especially at code review time. Organizations may consider incorporating explicit reward systems for developers who ensure security in the applications. For instance, the Vulnerability Reward Program [4] rewards any bug bounter who report vulnerabilities in Google-owned and Alphabet subsidiary web properties. Inspired by this initiative, companies and projects may create programs to reward developers who detect vulnerabilities through code review.

**Assumptions may hurt security.** Developers make security assumptions during code reviews. For example, Braz et al. [14] reported that reviewers justified not detecting vulnerabilities because they assumed the change author had already considered the application's security. In line with their findings, our study participants assume that security is the responsibility of another application's component or team. For instance, back-end developers reported assuming security as a front-end responsibility, while front end developers reported the opposite. However, developers need to be careful and spread awareness around these assumptions so that they do not end up hurting the application's security.

Some developers also thought their code is not security-sensitive because the software is intended for internal use. However, malicious and inadvertent activities may happen inside an organization. For example, acknowledging these scenarios, Microsoft has an insider risk management compliance solution that helps minimize internal risks by enabling clients to detect, investigate, and act on this type of activities [44]. Developers should be aware that internal code may still be vulnerable to malicious attackers that impersonate employees. Organizations may consider raising awareness on this issue and incorporating more strict software security policies into their development process to create a different attitude.

**Security can be learned.** Participants mainly reported insufficient security knowledge as a challenge when considering security during code review. Our results align with previous work [14, 16], in which developers reported lack of knowledge as the reason for not finding vulnerabilities in a code review task. These results strengthen the questions on the security education (or lack thereof) that developers are receiving. To create a different approach, educational institutions may introduce or give more attention to security in the software engineering undergraduate and graduate courses. For instance, in 2004, a team from the US Naval Postgraduate School won the "Capture the Flag" tournament at Defcon, the world's most popular hacker convention [22]. To learn security, students and developers must be able to switch from their traditional conditioning to the attacker's way of thinking [13]. Studies can be carried out to determine how to better educate students on software security.

Furthermore, to overcome developers' insufficient security knowledge, companies and projects may provide security training and allow time for learning. The free resources provided by OWASP [28] can be used to educate developers on security. For instance, organizations can adopt regular talks about the OWASP Top 10 List [52] to keep developers aware of the riskiest vulnerabilities, as well as training and seminar about new emerging vulnerabilities.

In addition, software developers can use Massively Open Online Courses (MOOCs) to learn security beyond the setting of their company/project. MOOCs are online courses that, additionally to traditional course materials, provide interactive courses with user forums or social media discussions to support community interactions among students, professors, and teaching assistants, as well as immediate feedback to quick quizzes and assignments [72]. Although previous research [65] has provided evidence that on-campus security courses still have better results than security MOOCs, developers can still benefit from them. For instance, our results show that companies allow developers to learn security during their work hours. These developers may follow MOOC activities from the office. Further studies can be conducted to investigate how to improve MOOCs to better support developers learning software security.

**A little help from the experts.** Most of our participants reported that they develop security-sensitive applications, but their code is rarely reviewed by security experts. This is a missed opportunity because code review can be used as a learning tool [8]. Including an expert in the review may not only increase the security of that specific change, but developers can make use of the knowledge sharing during the review to learn, *e.g.,* it can be a way to do onboarding in the security field.

The 2014 Cisco Annual Security Report estimated a potential shortfall of a million security professionals globally [18]. In 2022, the report of the World Economic Forum [27] stated: "There is an undersupply of cyber professionals — a gap of more than 3 million worldwide — who can provide cyber leadership, test and secure systems, and train people in digital hygiene." This way, even large companies do not have enough security experts to review every change made by developers. In fact, only one-third of our survey respondents reported having a security team in their companies/projects. As a solution, teams can select a developer to increase their security expertise and propagate their knowledge to other team members through code reviews. OWASP [28] suggests

that companies not only have a security team but also a further group of developers with an interest in security that can act as team local security Subject Matter Experts (SMEs) taking part in the code reviews [51]. Further studies can be designed and conducted to evaluate the effect of learning security in reviews.

**The right change to the right reviewer.** When not enough experts are available to review all code changes, the most security-critical changes can be selected and sent to the experts for a more scrutiny security analysis. Tools and processes can be developed to recognize security-critical changes and automatically assign security experts as reviewers. Previous research [9, 36, 53, 68] has investigated reviewer recommendation approaches, including recommendations based on cross-project work experience of potential reviewers and estimation of their expertise in specific technologies [53]. These approaches may be adapted to take security aspects into consideration. For example, they may incorporate security static analyzers, such as SonarQube [62] and Snyk Code [61], to measure the change's security-critical level.

**Code review is not perfect.** Code review is considered a valuable tool for finding defects and improving software quality [2]. In its current form [8], it requires short-term reactions from reviewers on the code changes [56]. In our studies, participants reported limitations specific to the code review process that hinder security assessment of code changes. For instance, developers reported that ensuring security is a time-consuming activity and do not have enough time for it due to their high workload. In addition, the security impact of code changes can spread in the application and not be limited only to the diff shown in the review. This way, to identify possible software vulnerabilities, developers need to have a good understanding of the application's components interaction, architecture design, and third-libraries usage, which does not always happen, thus often leaving software vulnerabilities undetected. Previous studies have reported the benefits of visualization techniques for developers [25, 35, 39], such as supporting them in better understanding or navigating the code. Future research can investigate whether developers can improve their understanding of the security impact of the code changes through the use help of code change visualization tools, such as Code Park [35] and CHANGEVIZ [31].

## 7 CONCLUSIONS

In this paper, we investigated the developer's perspective on assessing software security during contemporary code reviews, the support companies and projects provide to this process, and the challenges they face in this task. To this aim, we interviewed ten professional developers and surveyed 182 practitioners.

Our results show a change in the developer's attitude towards security – they acknowledge it is their responsibility to ensure security during code reviews. Yet, we also find that developers do not have security in mind when describing their code review practices, thus suggesting that security is not one of their main priorities when reviewing code. Moreover, developers make assumptions about the dynamic of the application for which they are reviewing code. These assumptions may lead them to disregard security aspects during their code reviews.

Our findings indicate that companies may need to change their approach towards security: They expect developers to ensure the security of their applications but provide little support and incentives to do so. Developers deny struggling to ensure security during code reviews. Yet, they report insufficient security knowledge, lack of knowledge regarding the application and its components, third-party libraries, human factors, and code review limitations as challenging factors that hinder security assessment at review time.

Our findings raise questions on the effectiveness of current methods employed in the software development process to ensure security, especially at code review time. To achieve secure and maintain secure applications, developers need to be educated about software security, and organizations need to improve how they motivate developers about it. Moreover, strategies to better support developers when ensuring security during code reviews should be further investigated to establish and improve the code review process.

## ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their thoughtful and important comments, which helped improving our paper. The authors gratefully acknowledge the support of the Swiss National Science Foundation through the SNSF Projects No. 200021M_205146 and PZ00P2_186090.

## REFERENCES

[1] 2020. GitLab: Mapping the DevSecOps Landscape - 2020 Survey. https://about.gitlab.com/developer-survey.
[2] A. Ackerman, L. Buchwald, and F. Lewski. 1989. Software inspections: an effective verification process. *IEEE Software* 6, 3 (1989), 31–36.
[3] A. Ackerman, P. Fowler, and R. Ebenau. 1984. Software Inspections and the Industrial Production of Software. In *Proceedings of the Symposium on Software Validation: Inspection-Testing-Verification-Alternatives*. 13–40.
[4] Google Alphabet. Last accessed in March 2022. Vulnerability Rewards Program. https://bughunters.google.com/.
[5] H. Assal and S. Chiasson. 2018. Security in the software development lifecycle. In *Proceedings of the symposium on usable privacy and security*. 281–296.
[6] N. Ayewah and W. Pugh. 2008. A report on a survey and study of static analysis users. In *Proceedings of the workshop on Defects in large software systems*. 1–5.
[7] N. Ayewah, W. Pugh, D. Hovemeyer, J. Morgenthaler, and J. Penix. 2008. Using static analysis to find bugs. *IEEE software* 25, 5 (2008), 22–29.
[8] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 712–721. https://doi.org/10.1109/ICSE.2013.6606617
[9] V. Balachandran. 2013. Reducing Human Effort and Improving Quality in Peer Code Reviews Using Automatic Static Analysis and Reviewer Recommendation. In *Proceedings of the International Conference on Software Engineering*. 931–940.
[10] D. Balfanz, G. Durfee, D. Smetters, and R. Grinter. 2004. In search of usable security: Five lessons from the field. *IEEE Security & Privacy* 2, 5 (2004), 19–24.
[11] T. Baum, O. Liskin, K. Niklas, and K. Schneider. 2016. Factors influencing code review processes in industry. In *Proceedings of the international symposium on foundations of software engineering*. 85–96.
[12] B. Boehm and V. Basili. 2001. Software Defect Reduction Top 10 List. 34, 1 (2001), 135–137.
[13] S. Bratus. 2007. What Hackers Learn that the Rest of Us Don't: Notes on Hacker Curriculum. *IEEE Security Privacy* 5, 4 (2007), 72–75.
[14] Larissa Braz, Christian Aeberhard, Gül Çalikli, and Alberto Bacchelli. 2022. Less is more: supporting developers in vulnerability detection during code review. In *Proceedings of the 44th International Conference on Software Engineering*. 1317–1329. https://doi.org/10.1145/3510003.3511560
[15] Larissa Braz and Alberto Bacchelli. 2022. Replication Package - "Software Security during Modern Code Review: The Developer's Perspective". https://zenodo.org/record/6875435. https://doi.org/10.5281/zenodo.6875435
[16] Larissa Braz, Enrico Fregnan, Gül Çalikli, and Alberto Bacchelli. 2021. Why Don't Developers Detect Improper Input Validation?'; DROP TABLE Papers;--. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 499–511. https://doi.org/10.1109/ICSE43902.2021.00054
[17] M. Christakis and C. Bird. 2016. What developers want and need from program analysis: an empirical study. In *Proceedings of the international conference on automated software engineering*. 332–343.
[18] Cisco. 2014. The Cisco 2014 Annual Security Reporte. http://www.cisco.com/web/offers/lp/2014-annual-securityreport/index.html.

[19] J. Cohen. 2010. Modern Code Review. In *Making Software*. O'Reilly, Chapter 18, 329–338.

[20] T. Cook and D. Campbell. 1979. *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Houghton Mifflin Company.

[21] A. Danilova, A. Naiakshina, S. Horstmann, and M. Smith. 2021. *Do You Really Code? Designing and Evaluating Screening Questions for Online Surveys with Programmers*. 537–548.

[22] Defcon. Last accessed in March 2022. Capture the Flag Competition. https://defcon.org/html/links/dc-ctf-history.html.

[23] K. Dempsey, P. Eavy, and G. Moore. 2017. *Automation Support for Security Control Assessments*. Technical Report. Technical Report NISTIR 8011, National Institute of Standards and Technology.

[24] A. Edmundson, B. Holtkamp, E. Rivera, M. Finifter, A. Mettler, and D. Wagner. 2013. An empirical study on the effectiveness of security code review. In *Proceedings of the International Symposium on Engineering Secure Software and Systems*. 197–212.

[25] S. Eick, J. Steffen, and E. Sumner. 1992. Seesoft-a tool for visualizing line oriented software statistics. *Transactions on Software Engineering* 18, 11 (1992), 957–968.

[26] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo. 2018. Empirical Software Engineering Experts on the Use of Students and Professionals in Experiments. *Empirical Software Engineering* 23, 1 (2018), 452–489.

[27] World Economic Forum. 2022. Global Risks Report 2022. https://www.weforum.org/reports/global-risks-report-2022/in-full/chapter-3-digital-dependencies-and-cyber-vulnerabilities/.

[28] The OWASP Foundation. Last accessed March 2022. OWASP Foundation. https://owasp.org/

[29] A. Furnham. 1986. Response bias, social desirability and dissimulation. *Personality and individual differences* 7, 3 (1986), 385–400.

[30] M. Galesic and M. Bosnjak. 2009. Effects of questionnaire length on participation and indicators of response quality in a web survey. *Public opinion quarterly* 73, 2 (2009), 349–360.

[31] Lorenzo Gasparini, Enrico Fregnan, Larissa Braz, Tobias Baum, and Alberto Bacchelli. 2021. ChangeViz: Enhancing the GitHub Pull Request Interface with Method Call Information. In *2021 Working Conference on Software Visualization (VISSOFT)*. IEEE, 115–119. https://doi.org/10.1109/VISSOFT52517.2021.00022

[32] N. Golafshani. 2003. Understanding reliability and validity in qualitative research. *The qualitative report* 8, 4 (2003), 597–607.

[33] M. Green and M. Smith. 2016. Developers are not the enemy!: The need for usable security apis. *IEEE Security & Privacy* 14, 5 (2016), 40–46.

[34] D. Hildum and R. Brown. 1956. Verbal reinforcement and interviewer bias. *The Journal of Abnormal and Social Psychology* 53, 1 (1956), 108.

[35] P. Khaloo, M. Maghoumi, E. Taranta, D. Bettner, and J. Laviola. 2017. Code park: A new 3d code visualization tool. In *Proceedings of the Working Conference on Software Visualization*. 43–53.

[36] Vladimir Kovalenko, Nava Tintarev, Evgeny Pasynkov, Christian Bird, and Alberto Bacchelli. 2018. Does reviewer recommendation help developers? *IEEE Transactions on Software Engineering* 46, 7 (2018), 710–731. https://doi.org/10.1109/TSE.2018.2868367

[37] T. Lindlof and B. Taylor. 2002. *Qualitative communication research methods*. Sage.

[38] S. Lipner. 2004. The trustworthy computing security development lifecycle. In *Proceedings of the Annual Computer Security Applications Conference*. 2–13.

[39] A. Mattila, P. Ihantola, T. Kilamo, A. Luoto, M. Nurminen, and H. Väätäjä. 2016. Software visualization today: Systematic literature review. In *Proceedings of the 20th International Academic Mindtrek Conference*. 262–271.

[40] McAfee. 2018. The Economic Impact of Cybercrime—No Slowing Down.

[41] G. McGraw. 2004. Software security. *IEEE Security Privacy* 2, 2 (2004), 80–83.

[42] A. Meneely and L. Williams. 2010. Strengthening the Empirical Analysis of the Relationship between Linus' Law and Software Security. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*. 1–10.

[43] A. Meneely and O. Williams. 2012. Interactive Churn Metrics: Socio-Technical Variants of Code Churn. *Software Engineering Notes* 37, 6 (2012), 1–6.

[44] Microsoft. Last accessed in March 2022. Insider risk management in Microsoft 365. https://docs.microsoft.com/en-us/microsoft-365/compliance/insider-risk-management?view=o365-worldwide.

[45] A. Nichols and J. Maner. 2008. The Good-Subject Effect: Investigating Participant Demand Characteristics. *Journal of General Psychology* 135, 2 (2008), 151–165.

[46] A. J Onwuegbuzie and N. Leech. 2007. Validity and qualitative research: An oxymoron? *Quality & quantity* 41, 2 (2007), 233–249.

[47] O. Pieczul, S. Foley, and M. Zurko. 2017. Developer-Centered Security and the Symmetry of Ignorance. In *Proceedings of the New Security Paradigms Workshop*. Association for Computing Machinery, 46–56.

[48] XM Platform. Last accessed March 2022. Qualtrics. https://www.qualtrics.com.

[49] A. Poller, L. Kocksch, S. Trpe, F. Epp, and K. Kinder-Kurlanda. 2017. Can security become a routine? A study of organizational change in an agile software development group. In *Proceedings of the conference on computer supported cooperative work and social computing*. 2489–2503.

[50] CWE Project. 2021. *CWE-327: Use of a Broken or Risky Cryptographic Algorithm*. Retrieved June 29, 2021 from https://cwe.mitre.org/data/definitions/327.html

[51] OWASP Project. 2017. *OWASP Code Review Guide 2.0*. Retrieved June 28, 2021 from https://owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2.pdf

[52] OWASP Project. 2017. *OWASP Top Ten*. Retrieved May 27, 2021 from https://owasp.org/www-project-top-ten

[53] M. Rahman, C. K. Roy, and J. Collins. 2016. CORRECT: Code Reviewer Recommendation in GitHub Based on Cross-Project and Technology Experience. In *Proceedings of the International Conference on Software Engineering Companion (ICSE-C)*. 222–231.

[54] P. Rigby and C. Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*. 202–212.

[55] P. Rigby, D. German, L. Cowen, and M. Storey. 2014. Peer review on open-source software projects: Parameters, statistical models, and theory. *Transactions on Software Engineering and Methodology* 23, 4 (2014), 1–33.

[56] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli. 2018. Modern code review: a case study at Google. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice*. 181–190.

[57] Y. Shin, A. Meneely, L. Williams, and J. Osborne. 2011. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *Transactions on Software Engineering* 37 (2011), 772–787.

[58] L. Sigelaman. 1981. Question-order effects on presidential popularity. *Public Opinion Quarterly* 45, 2 (1981), 199–207.

[59] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. Lipford. 2015. Questions developers ask while diagnosing potential security vulnerabilities with static analysis. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*. 248–259.

[60] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. Lipford. 2018. How developers diagnose potential security vulnerabilities with a static analysis tool. *Transactions on Software Engineering* 45, 9 (2018), 877–897.

[61] Snyk. Last accessed in March 2022. Snyk Code. https://snyk.io/product/snyk-code/.

[62] Sonar Source. Last accessed in March 2022. SonarQube. https://www.sonarqube.org/.

[63] D. Spencer. 2009. *Card sorting: Designing usable categories*. Rosenfeld Media.

[64] M. Tahaei and K. Vaniea. 2019. A survey on developer-centred security. In *Proceedings of the Symposium on Security and Privacy Workshops*. 129–138.

[65] C. Theisen, L. Williams, K. Oliver, and E. Murphy-Hill. 2016. Software security education at scale. In *Proceedings of the International Conference on Software Engineering Companion*. 346–355.

[66] T. Thomas, M. Tabassum, B. Chu, and H. Lipford. 2018. Security during application development: An application security expert perspective. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[67] C. Thompson and D. Wagner. 2017. A Large-Scale Study of Modern Code Review and Security in Open Source Projects. In *Proceedings of the International Conference on Predictive Models and Data Analytics in Software Engineering*. 83–92.

[68] P. Thongtanunam, C. Tantithamthavorn, R. Kula, N. Yoshida, H. Iida, and K. Matsumoto. 2015. Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review. In *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering*. 141–150.

[69] C. Weir, A. Rashid, and J. Noble. 2016. How to improve the security skills of mobile app developers? Comparing and contrasting expert views. In *Proceedings of the Symposium on Usable Privacy and Security*.

[70] R. Weiss. 1995. Learning from Strangers: The art and method of qualitative interview studies. *The Free Press* (1995).

[71] C. Wijayarathna and N. Arachchilage. 2018. Why Johnny Can't Store Passwords Securely? A Usability Evaluation of Bouncycastle Password Hashing. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering*. 205–210.

[72] Wikipedia. Last accessed in 2022. Massive open online course. https://en.wikipedia.org/wiki/Massive_open_online_course.

[73] I. Woon and A. Kankanhalli. 2007. Investigation of IS professionals' intention to practise secure development of applications. *International Journal of Human-Computer Studies* 65, 1 (2007), 29–41.

[74] Info World. 2004. Microsoft: More secure but mission not over. https://www.infoworld.com/article/2618608/microsoft--more-secure-but-mission-not-over.html.

[75] G. Wurster and P. Van Oorschot. 2008. The developer is the. In *Proceedings of the New Security Paradigms Workshop*. 89–97.

[76] S. Xiao, J. Witschey, and E. Murphy-Hill. 2014. Social influences on secure development tool adoption: why security tools spread. In *Proceedings of the Conference on Computer supported cooperative work and social computing*. 1095–1106.

[77] J. Xie, H. Lipford, and B. Chu. 2011. Why do programmers make security errors?. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*. 161–164.