SoK: Human, Organizational, and Technological Dimensions of Developers' Challenges in Engineering Secure Software

Azadeh Mokhberi
University of British Columbia
Vancouver, Canada
mokhberi@ece.ubc.ca

Konstantin Beznosov
University of British Columbia
Vancouver, Canada
beznosov@ece.ubc.ca

ABSTRACT

Despite all attempts to improve software security, vulnerabilities are still propagated within software. A growing body of research is looking into why developers are unable to develop secure software from the beginning. However, despite this attention, research efforts on developer challenges lack a coherent framework. We present a systematization of existing knowledge on the factors that make secure software development challenging for developers. We evaluated 126 papers to develop a framework of challenges that includes 17 areas of challenges in three dimensions of Human, Organizational, and Technological. These areas appear to influence each other directly and indirectly. Our work highlights the interplay of these areas and their consequences for secure software development. We discussed lessons learned from the framework, shed light on its role in assisting practitioners, and proposed directions for future research.

ACM Reference Format:
Azadeh Mokhberi and Konstantin Beznosov. 2021. SoK: Human, Organizational, and Technological Dimensions of Developers' Challenges in Engineering Secure Software. In *European Symposium on Usable Security (EuroUSEC '21), October 11–12, 2021, Online.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3481357.3481522

1 INTRODUCTION

Given that source code security vulnerabilities are the cause of many security problems [25, 49, 80, 82], developers can help prevent security vulnerabilities by writing more secure code. Developers are often blamed or considered "the weakest link", however Green and Smith state that "Developers are not the enemy" [40]. Developers play a critical role in security because bad decision-making and errors in coding have a huge impact on end-users and all those who rely on the developers' work [5, 40]. However, recent studies investigating the root causes leading to the vulnerabilities rather than blaming the developers have shown that developers face a variety of challenges that hinder them in engineering secure software. Researchers are now calling for more developer-centered approaches [5, 40, 66, 103] toward SSE to understand why developers are unable to develop more secure software. More importantly, we are facing a shift in developers' demographics. Software development used to be a niche for specially trained experts, but now everyone can be a developer [66] and make software available for millions of people. More than ever, we need more human-centered[1] [37] and user-centered approaches to research[2] to understand developers' needs.

The absence of a comprehensive framework of developers' challenges has motivated us to systematize knowledge around SSE. To the best of our knowledge, our research is the first of its kind to attempt to create a comprehensive picture of such challenges. The most relevant work is a survey of 49 papers by Tahaei and Vaniea [54], which focused on developer-centered security and classified methodologies and discussed themes emerging from the literature. Our paper presents the results of 126 papers and provides a framework of developers' challenges. Our paper reports a distinctive set of themes extracted from the studies not included in the previous survey paper (e.g., the role of developer personality). Another relevant work is a survey by Hamm et al. [41] that addresses methodological approaches in user studies in computer security.

Our paper was developed using classification, systematization, and organization. We evaluated the findings of 126 papers from 2007 to 2020 and classified developers' challenges and problems. We used the grounded theory [98] as the main method (§2) and then organized findings by using an affinity diagram [26]. The developed framework is shown in Fig 1. We built a framework of 17 areas of challenges that show the multiple dimensions of developers' problems in Human, Organizational, and Technological sections. These sections, directly and indirectly, influence each other. Some of these factors are more important than others because they affect many other factors (e.g., the role of managers, organizational culture, and attitudes and mindsets). In addition, our findings suggest that some qualities of SSE make SSE distinct from other types of software engineering.

Our paper contributes to current research in the following ways: (1) Performing the first systematization of knowledge on developers' challenges. (2) Identifying 17 developer-centered areas of challenges. These areas of challenges are the role of personality, attitudes and perceptions, deterrents, knowledge, experience, learning strategies, security culture, requirements and policies, support and strategies, structure and size, people in organization, interactions, APIs/libraries and protocols, analysis tools, updates and upgrades, online platforms, and languages. (3) Identifying the interplay among these challenge areas. (4) Discussing how having a framework of problems can assist practitioners and providing recommendations for how to use them. (5) Discussing avenues of future research.

EuroUSEC '21, October 11–12, 2021, Online
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8423-0/21/10.
https://doi.org/10.1145/3481357.3481522

[1] "Human-centered design is a group of methods and principles aimed at supporting the design of useful, usable, pleasurable and meaningful products or services for people" [81, p. 2].

[2] The main difference that distinguishes human-centered design and user-centered design from each other is the way they look at their target groups. In a user-centered approach, only users are the subject of study, whereas in human-centered design (e.g., participatory design, co-design, and co-creation) users are collaborators in research [74].

2 METHODOLOGY

With the help of literature reviews guidelines [83], this SoK (Systematization of Knowledge) paper systematically [61] organized published research papers. This systematization has been done based on grounded theory for review of the literature [98]. The process started with defining research criteria, determining resources, and searching publication repositories, as well as using a snowball approach [45] (illustrated in Fig. 1 in §7). Our search queries found 2846 papers in SSE, and we selected 126 papers published between 2007 and 2020, based on our exclusion criteria.

Inclusion and Exclusion Criteria: We included papers that met the following criteria: (1) must be a peer-reviewed journal article or conference paper; (2) must be about SSE; (3) must discuss developers' challenges in SSE; (4) could carry out user studies related to developers' studies related to SSE; (5) could discuss SSE about other stakeholders and present developer-related challenges (i.e., managers, testers, white-hat hackers, designers, security audits); and (6) could investigate developers' behavior (e.g., papers analyzing developer behavior on Stack Overflow). We excluded papers with the following characteristics: (1) no discussion about security aspects of technology; (2) no discussion of developers' challenges; (3) does not investigate security-related technology, (4) position papers and extended abstracts (we were only interested in original research); and (5) privacy-related papers.

Analysis: One researcher with expertise in UX research read all of the papers and extracted each paper's findings as direct codes. We qualitatively analyzed data from papers and used an affinity diagram [26], which is a powerful tool for identifying, analyzing, and acting on complex problems and large datasets. It is effective to visualize the emerged core categories [75]. This process identified three main categories. Next, together with a researcher who specializes in developer-centric security, the diagram was discussed and evaluated qualitatively. The affinity diagram results were further discussed with three other researchers. For any disagreement, authors listened to everyone's reasons and came to a consensus. Finally, three main themes with sub-themes were identified as the main developers' challenges in SSE (see table §8).

3 RESULTS

Based on our systematization, we developed a framework of 17 areas of challenges that can affect the development of secure software in three main dimensions of Human, Organizational, and Technological. The framework shows how human, organizational, and technological elements interact (Fig 1). A summary of our analysis is provided in table §8. This table indicates papers cited in each category of challenges.

3.1 Human

3.1.1 *Role of Personality.*

Both extremely high and low confidence[3] levels are linked to more vulnerable software. Developers who had never identified a vulnerability or consulted with a security expert had *less confidence* in their ability to develop secure software [84]. Unsurprisingly, *less confident* developers underestimated their ability to find vulnerabilities [86]. What is more, *overconfidence* led to developers' inability to protect themselves against further attacks or critically evaluate their own decisions [36]. Finally, confidence in the ability to detect security issues in Stack Overflow snippets was sometimes misplaced, which prevented developers from referring to external resources or formally validating the code [16].

Studies suggest an association between open-mindedness[4] and the abilities to spot and fix vulnerabilities and use security tools. It seems lack of open-mindedness can prevent developers from developing secure software. The results of a study based on a tabletop cybersecurity game [36] suggested that *open-mindedness*, together with adaptability to the game inputs, could make up for insufficient security experience. As such, those non-security experts who criticized themselves and their approaches frequently earned higher scores in the game [36]. Some developers tended to trust code blindly if it was from a reputable source like API documentation [62]. However, when subjects were open-minded, they were less likely to get trapped in API "blindspots"[5] where potential vulnerabilities can arise [63]. Also, in [96] researchers found a positive correlation between developers' inquisitiveness about security tools and their tendency to adopt such tools.

3.1.2 *Attitudes and Perceptions.*

Decisions and activities in organizations are guided by mindsets[6] surrounding security. Security can be seen as "a quality issue, a business value, a compliance obstacle, a technical challenge, a liability problem, a technical feature, or a sales pitch" [67, p. 2501], among other things. Such mindsets steer decisions on the selection and implementation of security activities and the ways security practices are incorporated into the routines of development teams in organizations [67].

Developers' security-related beliefs, mental biases, and perceptions could open doors for attackers. Some developers believed that they do not need to use security tools because they rely on "reviews or testing" by others in the same company [100]. Some developers believed that their code was secure, while researchers found vulnerabilities in it. The researchers noted that the worst-case scenario could happen when developers unintentionally released vulnerable code with confidence [6]. Also, some developers had an optimistic bias that could hinder SSE [9] (e.g., they thought that their applications were not an attractive target for attackers [9, 10]).

Developers find security hard to grasp. Developers considered security to be a complex concern [48, 102], as some information about it is "vague, contradictory, and sparse" [48, p. 37]. Also, there was no clear way to assess where there was "enough security" [47]. Developers believed that while their code had room for improvement, it was secure enough.

Although there are wide extremes[7] of security prioritization across all phases of the development process [9], the majority of the reported research points to security being at best a secondary

[3]Confidence: The quality or state of being certain [51] in one's knowledge or ability about developing secure software.

[4]Open-mindedness: "Openness relates to intellectual curiosity and the ability to use one's imagination" [63, p. 323].

[5]"An API security blindspot is a misconception, misunderstanding, or oversight on the part of the developer when using an API function, which leads to a violation of the recommended API usage protocol with possible introduction of security vulnerabilities" [63, p. 315].

[6]A mental attitude or inclination [53].

[7]We concentrate on developer challenges; therefore discussing when security was a top priority is out of the scope of this paper.

concern. Security was not a first-order concern in the eyes of developers [16, 90], not a priority in the developers' mindset [59, 62], and even remained a low priority among students (soon-to-be developers) [94]. Developers treated security as a secondary concern during design decision-making [4] and code development [62]. Security remains a secondary concern because of organizational factors. With tight deadlines [28, 102], limited budget [102], and the absence of value, reward, or monetary incentives [62] for SSE, developers did not invest much time and effort into security [94]. Also, the friction caused by other activities that developers usually engage in distracted them [67] and hindered the mental effort needed for security thinking [62]. Developers believed that both the security work and its connection with business goals was invisible [67]. Also, developers perceived that security efforts could interfere with business logic or complicate the software [65, 102].

Developers have perceptions toward security responsibilities that could be linked to less security effort, such as security being: (1) not their responsibility [16, 102]; (2) a shared responsibility [80, 100]; (3) the responsibility of all developers [10, 67]; (4) the responsibility of team members or of the organization as a whole [47]; and (5) a responsibility going even beyond the organization [47] (e.g., users or third-party security companies [23]). When some developers had a mindset that security was not a high priority or their responsibility, they did not use their security knowledge [9].

3.1.3 ***Deterrents.***

Deterrents[8] to engineering secure software are a human type of challenge that resulted from different factors from human, organizational, and technological dimensions. This section provides all such factors, showing how negatively these three dimensions interplay and influence human factors. A lack of motivation prevented developers from working on security. Not all developers had intrinsic motivation for security because it often fails to pique developers' interest in and of itself [67]. Some possible root causes include optimistic bias, the inability to identify any perceived loss [8, 10], and ambiguity as to whether effort spent on security, as part of a company routine, could result in concrete value [67].

In terms of human dimensions, personal attitudes, interests, or perceptions could deter developers from SSE. Some such deterrents at a personal level were seeing security as irrelevant to the developer's job [8, 10] or as others' responsibility, lack of interest in security, failure to see its value [8, 10], or a perceived conflict between proper and secure coding [8, 10]. Additionally, developer's perceptions, mentioned in the last paragraph in section 3.1.2, acted as a deterrent.

In organizational dimensions, a lack of planning and support, resource shortages, and a lack of knowledge can lead to developer discouragement. The first group of deterrents in organizations includes "competing priorities and no plan," lack of organizational support [8–10], and lack of rules provided by government regulations and organizational policies [102]. The second group includes resource shortages such as budget, time, and staff [8–10]. The third group includes a lack of expertise [8–10] and awareness about security tool availability [8, 10]. The absence of formal security plans and unfamiliarity with code analysis tools are the first and second most frequently occurring deterrents among developers in organizations of various sizes [10]. However, the first and third groups were more frequent in smaller companies [10].

[8]"Deterrent: serving to discourage, prevent, or inhibit" [52].

In terms of technological dimensions, security tools' poor usability, high complexity, lack of integration with the development environment, and high false-positive rates turn developers away from automating security analysis of their code. Too many false positives were very discouraging for some developers [96], who might stop using analysis tools [44, 80]. Other deterrents that demotivated developers from using static analysis tools were "disjoint process": having to go outside of a code development environment to use a tool or see the tool's results [44], tools' interface complexity [96], perceived uselessness of tools, and lack of awareness of the tools [10]. Also, the ways warnings were presented dissatisfied developers [44] (see section 3.3).

3.1.4 ***Knowledge.***

Developers' knowledge and willingness to seek knowledge are important factors for SSE. Security knowledge is linked to the developers' mindset and behavior. Insufficient security knowledge is correlated with the inability of developers to apply security measures, implement these measures, and understand the security implications of their choices. Developers' unwillingness to learn [90] and keep their knowledge up-to-date [58] were issues when it comes to promoting security awareness. Also, many developers had no security-related education and sought more knowledge when they faced security issues at work [17]. In [59], having security knowledge by participants did not mean that this knowledge was applied. Participants with little or no security understanding, on the other hand, were able to come up with secure implementations.

In many small companies, insufficient security knowledge among developers is linked to their inability to engineer secure software. In the wide spectrum of developers' knowledge, some developers did not have new or updated knowledge [50] and, in some companies, there were no expectations for developers to have security knowledge [9]. Those who worked in small companies tended to have less knowledge about secure coding than their counterparts in medium to large companies [100]. Due to not having enough security knowledge, developers were unable to implement security measures [23] or, worse, cases of misunderstanding had far more impact on making code less secure than simple mistakes [85], and developers could not reliably engineer secure software [9].

A lack of security knowledge can also adversely influence developers' mindsets, behaviors, and actions. Lack of knowledge and false confidence were reasons for some developers mistakenly believing that their code was secure and being unable to recognize vulnerabilities in it [95]. Even if developers were able to implement some security measures, they failed to understand the threats [93], disregarded the value of implementing security measures [8], or employed "lax security practices"[9] [9]. When developers were unable to understand the security implications of different implementation and design options, they got frustrated and chose easy but insecure actions [50]. Finally, developers' lack of security knowledge prevented them from properly evaluating security-related code snippets from Stack Overflow [16].

Knowledge of security and security analysis tools is correlated with their usage. Developers had different levels of awareness of

[9]"Security is not considered in the design stage, Security is not a priority during implementation, Developers do not test for security, and Security is not considered during code review" [9, p. 291].

analysis tools. Some developers used them without a complete understanding of tool functionality [9]. Developers' lack of security knowledge could result in tool misuse, which led to insecure solutions [95]. The number of times program analysis tools were used was significantly correlated with developers' familiarity with them as well as their amount of focus on security [24], which means better understanding of analysis tools led to higher adoption rates and trust in the tools' abilities to find bugs and vice versa [24].

3.1.5 *Experience*.

The correlation between years of work and coding experience and SSE is still unclear. In [31, 63], the amount of programming experience a developer had does not correlate with improved accuracy or efficacy when it came to discovering security vulnerabilities in code. Another study found that professional developers wrote code that was, while more functional, not necessarily more secure than code produced by (less experienced) students [4]. However, the samples of these studies were not representative for generalization, so this finding requires more investigation (see §4). The appearance of different types of security vulnerabilities did not seem to correlate significantly with either developers' experience in years or in completed security training [63, 85].

Although experience seems not to correlate with tool usage, specific experience with tools showed a positive correlation with developing secure software. While a research team reported a link between the amount of experience in software development and the adoption of security tools after their qualitative study [96], they could not confirm the correlation in a follow-up survey [97]. They found that experienced and inexperienced developers did not differ in terms of using security tools [97]. For achieving the best outcomes of a static analysis tool (SAT), it seems essential to have experience with tools besides general security experience [15].

Having security experience could increase self-efficacy, long-term awareness about security, and use of security tools. More experience with finding vulnerabilities or working with security experts led to increased "secure-development self-efficacy" among developers [84]. Experiencing real security issues, e.g., a security breach, increased developers' awareness and concerns about security in the long run [10]. Also, experiences of having their systems hacked pointed developers toward using security tools [100].

The correlation between experience with programming and the ability to write secure code remains unclear. On one hand, a survey of 307 GitHub users [6] found that when security background was not a significant factor, years of experience in programming Python was a significant factor in producing more secure code. On the other hand, in a mixed-methods field study with 43 freelancers [58] and a replication of it [27], the difference in the years of Java experience was not a significant factor.

3.1.6 *Learning Strategies*.

Developers can take various paths to obtain security knowledge, but can struggle to make connections between learned knowledge and the current security problems they face. While a traditional way to learn about security is taking security-related courses [17, 57, 62], this path does not appear to be the most common. Out-of-class education might include reading research papers [42], attending conferences [17], attending corporate training [17, 57], or certification [17]. Specifically for learning about cryptography, educational approaches include being involved in crypto standards groups [42], taking Coursera courses [56], and reading books [56]. When developers have formal security training, they were more likely to spot security-related issues in the code (known as "blindspots") [62]. Nevertheless, it was difficult for developers to find correlations between theorized vulnerabilities learned about in school and the security problem at hand [62]. Additionally, in [59] even when developers intended to save passwords securely, they usually did it insecurely due to outdated methods they learned. Despite the fact that security is a rapidly changing field, participants did not keep their knowledge up to date.

3.2 Organizational

3.2.1 *Security Culture*.

Security culture[10] in an organization can contribute to SSE. in a study of three large companies,[11] the companies ranged from those with a "security first" corporate culture to others where security wasn't always the primary priority for each business unit [20]. One of the most important deciding factors in encouraging security in development and motivating developers to use new security tools was determined to be the company's security culture [97, 100, 102]. Mature, strong cultures around security led to more thorough SSE and testing processes [42]. Moreover, such a culture was linked to shifting developers' attitudes toward considering security to be a shared responsibility [9]. However, the culture in some companies discouraged developers from buying new security tools via a lengthy authorization process [100]. In [9], the security-inattentive group,[12] had a company culture that typically did not include security, and as a result, developers frequently assigned security responsibility to a single person or team.

Developing a security culture requires internal and external motivations. The internal motivations include beliefs about the importance of security [42], and the external motivations include gaining a larger market share or requirements from the customer [42]. Having security-related workshops [7, 46, 67, 88] can built such motivations. However, as long as security practices are not part of the "ostensive and performative aspects of organizations,"[13] workshops' effects could be short-term and not lead to security practices becoming routines at the organization [67].

3.2.2 *Requirements and Policies*.

Lack of clear security-related task descriptions could lead to less security work. For some developers, when responsibilities and roles defined in the teams had conflicts with applying security, they did not follow best practices [9]. Developers paid attention to security only if asked to do so [101, 102] by either the design requirements [101], their clientele [18, 57, 58, 101], their superiors' expectations [97], or regulations [101]. Developers in [28] reported that they faced lack of general security guidelines and nobody in the company was in charge of ensuring that these security requirements were followed.

While some factors help developers to see security needs, developers reported a lack of visibility. Security is a quality requirement,

[10]"The beliefs and social norms surrounding security and security tools in a company" [100, p. 1100]

[11]14,000 and 300,000 employees

[12]"The group who barely considered security or did not consider it at all" [9, p. 283]

[13]"The ostensive aspect of a routine embodies what we typically think of as the structure. The performative aspect embodies the specific actions, by specific people, at specific times and places, that bring the routine to life. [34, p. 94]

which makes it invisible in agreements between managers and developers. Elements such as external triggers (e.g., customer feedback) made security more visible [67]. Developers, regardless of their knowledge about security, were reluctant to apply their knowledge when security was invisible; therefore, it is required to explicitly request security [59]. Assal et. al, [9] suggested that security needs to be included in company culture and policies, and promoted from the higher echelons of the organization. Seeing a higher need for security was a reason developers showed a tendency to use more security assurance and run updates more often [89].

Code can be made more secure through company policies. Adopting concrete security policies increased code security [27] as shown also in [100]. Security tools were employed by all participants who were required to use them. Policies such as limiting tool training to a specific group in the team negatively affected developers' mindsets because it made them feel less responsible for secure software [100]. In addition, developers reported "security standards are informal, verbal 'best practices' [100, p. 1099]" in their organization. For instance, in some organizations there were no official policies on using security tools or reviewing warnings in a timely manner [13].

3.2.3 ***Support and Strategies.*** Lack of organizational necessary resources plays a crucial role in preventing developers from SSE. Examples of resources included budget [9], external penetration testers [9], knowledgeable security experts [9, 80], and security tools [100]. Another required support was budgeting time to do proper security development and testing while not delaying the product's release date [42]. Lack of such support could lead to violating best practices and increasing the security team's burden significantly. As a result, there would be more chances of security vulnerabilities as well as the loss of valuable employees on the security team [9]. Furthermore, lack of resources might be the reason developers thought their applications were vulnerable in spite of satisfaction with their practices [10].

Security work is different in each development stage. In the security development life cycle, more security-related work is done in the implementation stage in comparison with analysis, test, review, and post-development test phases. Also, more security work is done in the design stage compared to analysis and review ones [10]. In the design phase, there is a shortage of security experts [21]. Security considerations in the design stage can be ad hoc, as developers (non-security experts) are responsible for them, and if they fail to pick security-sensitive features, they might be ignored [9, 21].

3.2.4 ***Structure and Size.*** Integrating security into the organizational structure is a useful strategy in SSE. If security is not built into the structure of the organization, it could fail to be integrated into organizational routines [67]. The success of integrating security work in organizational routines depended entirely on the perceptions that people in an organization held toward security [67]. Organizational structure affected whether developers felt responsible for security, with developers who knew there were experts in their organization specifically responsible for software security having a relaxed attitude toward security [102].

Security implementation motivation appears to be linked to the size of the company. Company size correlated with developers' motivations [10]. In smaller companies, developers were less motivated to implement security because they were not prepared for security work and lacked a plan to strike a balance between different priorities [10]. In addition, developers in smaller companies relied on social networks and search engines to look up information related to developing secure software, whereas in larger companies developers could seek answers from security experts at their company [17]. During the software development life cycle, the team context might have an impact on the code review process [28].

3.2.5 ***Role of Individuals in Organization*** . Individuals play a role in influencing the routines and activities inside the company.

Managers' mindsets, behaviors, and decisions correlated with developers' behavior and decision-making. Managerial roles were important for several reasons: (1) Their decisions affected the final product, because they could exercise discretion to approve a release despite the presence of security vulnerabilities [80]. (2) What was eventually built is what managers told developers to build, even if developers had an interest in putting more effort into security [9]. For instance, economic considerations around product development could result in security flaws being sometimes intentionally added and/or overlooked to satisfy stakeholder requirements [65], or sometimes, managers cared about security aspects when "negative publicity [arose] from an application compromise" [99]. (3) Developers looked at what their managers expected them to do, sharing their mindset about security [67]. For instance, security is a non-functional requirement that managers still considered an implicit quality along with usability and maintainability—a mindset imitated by developers [67]. Without explicitly mentioning security, managers expected development teams to build *secure* software [67]. (4) They helped form a security culture and policies to foster such a culture (e.g., encouraged developers and security experts to collaborate, or offered security-related classes [100]).

Peers, experts, and champions influence code security. Recommendations of new software tools by peers and managers were more effective and trustworthy than other information sources [97]. Also, security expert or champion involvement in project teams resulted in a greater number of crypto API defects, likely due to teams using more cryptography [89]. The number of these issues is not a result of security requirements or assurance technique usage [89]. The involvement of security experts/champions is a reason developers used more security assurance and ran updates more frequently [89].

3.2.6 ***Interactions.*** Developers face challenges in interacting and communicating with others in the organization. For instance, there is a lack of effective communication between different stakeholders in product teams responsible for designing security features, which could lead to more tension [21].

Developer and security expert interaction could affect the former's attitudes, tool adoption, and decision-making. Fixing vulnerabilities was not just a matter of having security knowledge or working harder on the part of developers. Instead, new ways of collaboration between developers and security experts were needed [65]. Interactions with security experts (1) caused developers to feel more responsible for the security aspects of their work [97, 100] due to more social pressure from the security team as the latter reviewed and audited their code [100], (2) increased developers' interest in using security tools [97], and (3) facilitated decision-making around security with regards to costs, tools, and time, among other things [90].

However, these interactions were challenging and needed to be facilitated [80, 92].

Interactions in organizations are associated with many conflicts and coordination challenges. Conflicts between auditors and developers could result in leaving vulnerabilities in the software [80]. When fixing vulnerabilities required the approval of several teams of stakeholders like developers, managers, etc., (e.g., convincing developers to see the real harms and serious vulnerabilities in the software) [80], the teams' diverging interests could lead to vulnerability fixes being delayed. Communication with developers about detected vulnerabilities in their code was challenging due to different findings: (1) security directors or consultants needed to find effective means of communication to motivate developers to firstly acknowledge and secondly fix vulnerabilities [80], (2) security experts needed to be mindful of developers' feelings when pointing out security vulnerabilities in their code [80], (3) hackers and testers thought they should be open-minded and adopt a respectful tone with developers to prevent them from becoming defensive [86].

3.3 Technological

3.3.1 ***Security APIs/Libraries and Protocols.*** APIs/libraries[14] increase developer efficiency and promote code reuse. However, the misuse of such tools can decrease the security of software (e.g., misuse in crypto APIs is a main cause of leaving Android applications vulnerable [32]).

The complexity of crypto APIs could lead to their misuse, and this challenge was compounded by poor API documentation. The complexity of such APIs (e.g., absence of guidance for selecting parameter values) sometimes made them too hard for developers to learn and use [95]. For instance, APIs' ambiguous underlying implementations hindered developers' ability to use the "sequence of method calls & parameters" in the correct order [56] and assign values to parameters to use APIs [95]. Developers had difficulties identifying what API to use and setting up the environment [56], faced inappropriate levels of API abstraction (e.g., too low-level) [56], and encountered poor documentation [2, 56]. Developers faced a lack of guidance to prevent API misuse (e.g., use of a constant string as the "salt") [95], missing code examples [2], poorly designed APIs (e.g., misleading defaults and difficult debugging) [56], and a lack of auxiliary features (e.g., secure key storage) [2]. The difficulties mentioned above pushed developers to use potentially unreliable online resources such as Stack Overflow or blogs [2]; these resources often suggest insecure workarounds [2, 50]. We must consider that developers tend to choose more usable APIs even if they are less secure [95].

Developers' difficulties with TLS result in security issues. Developers sometimes customized apps by deviating from the default TLS certificate validation policies that are secure by default, which resulted in a significant reduction of security [33]. Many developers did not realize the dangers around insecure TLS connections. The root causes seemed to be a lack of understanding of how and what functions a TLS serves, a lack of usable TLS warning messages, and poor support for self-signed certificates [33].

[14]"API" and "library" are often used interchangeably for the purpose of this discussion.

3.3.2 ***Analysis Tools.*** Assistive tools facilitate finding security vulnerabilities by saving time and effort in comparison with manually looking for bugs [44]. Most users of security tools are from large companies [100]. Google [12, 13, 71, 72], Microsoft [24], and Facebook [30] use static analysis tools (SATs) in the various stages of software/app development. SATs, which are most often used to find security issues [24], are associated with many difficulties for developers.

Poor learnability and high complexity of SATs hinder their adoption by developers. Developers deemed security tools to be too complex [96, 97] and suitable only for people who have security expertise [79, 96]. Poor learnability of a security analysis tool affects its adoption [96]. For instance, the biggest pain point for participants in an study [24] was the mismatch between default settings and developer needs.

Developers have difficulties finding a proper tool or incorporating it into their own workflow. Those who worked with dynamic languages believed there were no good analysis tools for such languages [96]. In addition, it was difficult to find tools with support for all relevant languages and frameworks being used in an organization [80]. Moreover, using tools required a "disjoint process" which did not fit into developers' routine coding behaviors [24, 44, 76]. A lack of support for progress tracking and batch processing interfered with developers' workflows [76].

Other challenges with using analysis tools are high cognitive demand, insufficient provided details, and a lack of trust in the tools. To fix vulnerabilities detected by the SATs, developers had to perform large numbers of cognitively demanding tasks [77]. Besides that, developers needed more information regarding potential vulnerabilities [76, 77] or else they had difficulty believing that reported vulnerabilities were real, exploitable problems [77]. Even more, some developers did not trust analysis tools with locating security issues [9] or did not think such tools could find complicated issues, such as reliability-related ones [24]. Developers need to receive quicker feedback in real time [101] as long as this does not disturb development workflow [44].

Analysis tools often require too much time, effort, knowledge, or resources, which frustrates developers. Such tools were often viewed as too costly [96], slow [24, 96] (e.g., slow performance in large applications in modern static and dynamic analysis tools) [80], or time-consuming [13, 79] (during installation, configuration, running, and reading output [96]). Reviewing warnings, especially when running a SAT for the first time, used too much time and effort [11, 13].

Warning messages from SATs assist developers in identifying and addressing vulnerabilities, especially when the warnings are ranked and prioritized by the SATs. Many developers fixed high-priority, severe, new, and correctness-related warnings [12, 13]). But the relevancy of low-priority warnings depended on the users' context. In addition, some elements such as project and/or organizational characteristics (e.g., the amount of time they would prefer to invest) affected developers' likelihood of reviewing and fixing flagged issues. Another pain point for developers was the lack of proper ranking of warnings [24] or a lack of vulnerability severity and priority score information [76]. Developers needed to give priority to warnings about security and best practices [24], and it was important for them to have the ability to suppress [24, 28, 44] or filter some warnings [43].

The prevalence of false positives appears to be the most important usability pain point. SATs' false-positive warnings were widely cited as the main challenge for developers [9, 24, 43, 44, 80, 96]. Therefore, reducing false positives [43], along with excessive warnings, appears to be the main avenue for increasing SAT adoption.

Poor quality and usability issues of SAT warnings also frustrate developers. Developers considered warnings in analysis tools to be poor in quality [24, 43, 44, 77]. This could be the result of (1) weak visualizations and complexity of user interfaces [24], (2) poor interface scalability in showing vulnerability sorting, vulnerabilities' overlap, and visualization issues for large amounts of calls [76], (3) overloading of IDE interfaces [28], (4) lack of adequate affordances for assisting developers in navigating and managing reported issues, (5) inaccurate analysis [76], and (6) irrelevance [9]. A recent study showed that due to developer heterogeneity, there is no one warning type that is preferable for all developers [28]. For instance, pop-ups were disliked by participants with more programming experience, but they were liked by those with less programming experience.

Developers have difficulties understanding, interpreting, and addressing warnings from analysis tools. Although some developers tended to follow fix suggestions [77], tools did not provide enough information about the nature of the issue [44, 76], the reason behind it [44, 76], and how it could be fixed [24, 44, 76]. Tool outputs were hard to read even for more experienced developers [15]. In addition to having problems with understanding the results [96], the way SATs described an issue might differ from how developers perceived it. When provided with a better explanation, however, developers could better understand the problem [43]. Moreover, developers found it difficult to locate resources and documentation even when links to outside material were provided, whether because the latter had not been described adequately or the developers did not have confidence in the material [76, 77]. Lastly, mismatched examples and line numbers confused developers about the results and finding the vulnerabilities [76].

3.3.3 ***Online Platforms****.* Previous studies showed that online platforms are being used widely by developers. However, recent findings showed that code examples in Q&A platforms may contain security vulnerabilities [1, 3, 22, 35]. Therefore, it is vital to understand challenges developers face in using such technologies.

Developers turn to online platforms when facing problems. They spend time reading online resources [56] and looking for online advice [17] (e.g., OWASP was listed as a trustworthy platform for security tasks [90]). An international community of developers has formed as a result of Q&A[15] platforms, and such interactions have had a visible impact on software development practices around the world [48]. Stack Overflow [64] and search engines were two major resources for developers to find solutions for security-related problems [3]. Developers copied and pasted code from external resources, of which Stack Overflow is the most common [35]. The fact that Google, as the most popular search engine, put online resources like Stack Overflow before official documentation encouraged developers to review unofficial documents before official ones [95]. However, it is not straightforward to determine whether Stack Overflow causes more harm than good, since tracing vulnerable code to Stack Overflow is not a surefire thing [35].

On the negative side, Stack Overflow contains many insecure solutions that can impact SSE. Although it seems developers do not accept all the provided information on Stack Overflow without question [48], it can still introduce vulnerabilities into developers' code. Some posts on Stack Overflow contain incorrect information and cause errors in the code developers borrow from it, which could negatively impact SSE [3, 35, 50]. App developers who used Stack Overflow as an information resource produced significantly less secure code in comparison to those who used other sources (e.g., books and official documentation). However, developers who used Android documentation had much less functional code in comparison with developers who used Stack Overflow [3]. Also, on Stack Overflow, while security-related questions were popular among developers, the Stack Overflow community often failed to provide satisfying answers for programmers [69] in this area. Specifically, in [35], the authors found that applications that used code snippets from Stack Overflow contained, at the minimum, one vulnerable code snippet. Another study showed that more than one-sixth of the apps analyzed used code from Stack Overflow, of which the majority included at least one insecure snippet [35]. Seekers could be directed to and influenced by wrong answers simply because some posts from popular responders receive more attention than others [50]. Finally, many Stack Overflow code examples reviewed in [104] assumed a level of knowledge and lacked complete explanation of the problem (particularly for novice developers).

3.3.4 ***Updates and Upgrades****.* Using third-party libraries could impact the security of the developed software [73]. Updates and upgrades of these libraries play an important role in keeping software secure. While finding security bugs acted as a motive for developers to update their products [29, 73], they found it challenging to perform updates of third-party cryptographic libraries [42].

Developers' concerns prevent them from keeping up with updates. Reasons for not updating or upgrading libraries included (1) a cost-benefit trade-off, (2) a lack of incentive when the app already worked as intended [29], (3) the fear of breaking an already-working app [73], (4) the fear of triggering Federal Information Processing Standards Publication recertification [29, 42], (5) unfamiliarity with advantages that updates bring [29], and (6) the belief that as long as developers or users do not experience any problems, changes are not needed [73]. Additionally, developers mentioned that there was a lack of clarity on whose responsibility it was to upgrade the product after the release [90].

3.3.5 ***Languages****.* More research is needed to better understand the challenges that developers have with languages in SSE. The result of a contest with 116 teams [70] showed that the best submitted works used C/C++. Statically typed language submissions were less likely to have security problems, and teams using a diverse range of programming languages wrote code that was more secure. In another contest [68], no definitive connection could be established between a particular language (Java, PHP, Perl) and the security of the resulting code; rather, the way a platform was used trumped the nature of the platform itself [68]. Nonetheless, a later study found a higher injection rate of vulnerabilities in applications written in PHP than

[15]question and answer

in other programming languages [19]. In another study, researchers could not find a meaningful difference between the use of Spring or JSF as a framework to develop more secure software [27, 58].

4 DISCUSSION

4.1 Limitations

It is important to note that even though our intention has been to take as many developer problems into consideration as possible, we do not claim we have not missed some. We remind the reader again that the set we have presented here should be viewed as a beginning for a new comprehensive way of thinking about systematization of knowledge in developer centered security to be perfected by future researchers. Additionally, we have not provided our assessment of the generalizability of the results of the cited studies given the demographics of the samples.

4.2 Developed Framework

In Fig 1, we depicted a comprehensive picture of challenges and their interrelationships. By systematizing the current literature, we have found that SSE is affected by many factors that can be categorized into Human, Organizational, and Technological. These factors appear to influence each other directly and indirectly. According to our systematization, organizational dimensions have the most interplay with other areas (see the arrows in Fig 1). Among areas of challenges, this figure shows that analysis tools, attitude and perceptions, knowledge, and deterrents, respectively, are the top challenges affected by other challenges. It seems that role of individuals, requirements and policies, knowledge, attitudes, and organizational culture, in that order, have the greatest influence on the other challenges. This Fig 1 is the current landscape of challenges found by researchers. Although this does not mean that the relationships not drawn do not exist (e.g., Discussed in [38] about APIs' interactions with other technological factors).

This framework would help to find some patterns for future research directions. We found that some areas of challenges play a more important role because they have effects on many other factors. To illustrate: (1) managers play important roles by forming organizational culture, enforcing policies, influencing developers' mindsets by sharing their mindset and decision-making, and setting priorities (section 3.2.5, section 3.1.2); (2) organizations' culture, developers' knowledge, and experience influence developers' attitudes and perceptions (section 3.1.5, section 3.2.1, section 3.1.2); (3) attitudes and perceptions act as motives or deterrents for developers (section 3.1.3); (4) knowledge of security and tools is associated with the use of code analysis tools (section 3.1.5, section 3.3.2); (5) organizational factors, such as policies and culture, are linked to developers' knowledge, tool adoption, and attitudes toward secure software development (section 3.2.2, section 3.2.1).

This framework would help to find potential spots where developers could be hindered by challenges. The framework shows some interactions between different areas, which represent interesting findings that need future work. For instance, there is a loop between online platforms, learning strategies, and knowledge,[16] as discussed in section 3.1.6 concerning the learning strategies that influence developers' knowledge. Also, the knowledge correlates with how developers use online platforms. As noted in section 3.3.3, developers used online platforms to gain knowledge. Observing such loops (Fig 2) could explain why some measures are not effective when developers are trapped in such loops, the security of the software could decrease; on the other hand, this could be an opportunity to help developers to improve the security of their software.

Soon after we finished our review, we found a couple of new research expanding developers challenges (e.g., [55, 78, 87]) that discussed new challenges in this area. Therefore, future studies can focus on the following: (1) Challenges that had not been identified at the time of writing this paper. (2) New interplays between challenge areas, such as learning strategies and the role of personalities. (3) Loops in challenges and how they can help with developing secure software and how they influence developers' security-related tasks.

Building an integrated framework of challenges would benefit academics in a variety of ways. Our findings suggest that some qualities of SSE make this area distinct from other areas in software engineering. These qualities include security being invisible, having a lower priority than functionality, and potentially requiring more motivations. Additionally, there are more obstacles due to constant changes in the security landscape and threats [91], lack of access to security expertise [89], and a lack of documentation or information section 3.3 in comparison to functional coding. The developed framework in this SoK aids researchers in identifying potential areas for further research. Based on 17 identified factors and their interplay in the previous section, we provide some examples of possible directions and recommendations to practitioners and suggest research directions for academic researchers.

Practitioners would benefit from this systematization of knowledge in two ways: (1) To gain a better understanding of the multi-dimensional challenges that developers face while designing secure software, as well as how these aspects interact. (2) To assist practitioners in understanding the big picture of difficulties while developing new policies and strategies to improve software security. The list below outlines a theoretically ideal state to strive for according to the best effort principle. All of these are suggestions and could be research questions for future work.

4.3 Recommendations for Practitioners

Visibility of secure coding: Make security a visible aspect of the software to encourage developers' use of knowledge and sense of responsibility. By visibility we mean measures such as making it an obligatory part of the development process, transparently discussing the need for security, budgeting enough time for the work, clearly assigning developers to security-related tasks, and calling for security-related meetings. Regardless of their knowledge of security, developers are reluctant to apply their knowledge when security is "invisible" (section 3.2.2, section 3.1.2). Therefore, *developers should be provided with more visible incentives for SSE.* Another problem created by the lack of visibility is having outdated software due to a lack of clarity on who is responsible for upgrading the product (section 3.3.4). Therefore, *more visibility is needed about who is responsible for upgrading the product after release*. Finally, shortage of time is an obstacle for developers, acting as a deterrent for SSE

[16]A loop occurs when relationships between challenges (visible in Fig 2 as arrows between nodes) form a cycle.

(section 3.1.2, section 3.1.3, section 3.2.3). *More visibility is needed for explicitly allocating time in project budgets for engineering more secure software.*

Policy/Culture/Strategies: To have secure software, organizational policy, culture, and strategies can assist developers. Favorable organizational structure, policies, and culture, and concrete plans for security have been shown to assist developers in SSE (section 3.1.3, section 3.2). *Organizations need to consider SSE in designing each of these aspects of the organization.* Lack of sense of responsibility for software security correlates with less SSE, whereas professionally responsible developers are motivated to increase their expertise, awareness, and completion of security-related tasks (section 3.2.4, section 3.1.3, section 3.1.2). *It should be made clear to developers that security is their responsibility even if they have dedicated "security experts" in their team or organization.*

Communication and collaborations with other stakeholders in an organization can lead to more SSE. Poor communication between developers and security directors, consultants, or experts can delay fixing vulnerabilities (section 3.2.6). Therefore, *promote a culture in which developers feel safe acknowledging, reporting, and fixing security vulnerabilities in their code.* Additionally, according to the literature, the support provided by other people or interactions with them in the chain of SSE is associated with the promotion of security mindsets, increases in developers' self-efficacy, learning about vulnerabilities, improvements in developers' understanding of vulnerabilities, more skilled use of tools, more motivation, and overall more secure code (section 3.2.6, section 3.2.5, section 3.1.3, section 3.1.6). *Organizational strategies can facilitate collaborations and interactions between developers and security experts*.

Capitalize on specific personality traits. Research suggests that people with a high degree of curiosity or open-mindedness tend to outperform others in secure software development (section 3.1.1). Therefore, *paying attention to these traits when recruiting software developers will amplify efforts to improve software security.* Also, open-mindedness can make up for insufficient security experience and would help to avoid being trapped in APIs' blindspots (section 3.2.6, section 3.1.1). *Cultures could be promoted in which developers practice or engage in open-mindedness activities to think critically about their work and be open to vulnerabilities reported by others.*

Employ organizational factors to support human factors in improving SSE. Attitudes, mindsets, and perceptions steer developers' decision-making and actions toward engineering more secure software (section 3.1.2, section 3.1.3). To this end, *organizations can develop strategies to facilitate the cultivation of security mindsets among developers* through training, culture, policies, etc. Moreover, motivations and deterrents count as important factors that influence other factors, such as attitudes and mindsets, use of tools, and forming security culture. In this realm, intrinsic motivations seem to have the upper hand in swaying developers toward SSE (section 3.1.3, section 3.2.1). It is in the hands of the organizations to *promote and/or facilitate the development of intrinsic motives toward security* (e.g., enjoying doing security-related tasks).

Finally, research suggests that a better understanding of analysis tools leads to higher adoption rates and trust in the tools' abilities to find bugs. Considering all benefits of using analysis tools, such as facilitating finding security vulnerabilities, security-centered policies could reduce the barriers caused by such tools (section 3.3.2, section 3.1.5, section 3.2.2). Companies should design policies that make developers more familiar with analysis tools and encourage more usage. For example, companies can make tools available for all developers, regardless of their professional expertise, or reward developers who help others in using these tools.

Knowledge and learning: Encourage developers to improve their knowledge and experience. Having security knowledge correlates with tool adoption. Shortage of knowledge with the inability to implement security measures, mistakenly believing the code is secure, and lack of understanding of threats finally make developers frustrated so that they may choose easy but insecure options, e.g., copying and pasting insecure code from Stack Overflow (section 3.1.5, section 3.3.3). *If the organization hires developers without security knowledge, developers should be provided with opportunities and incentives to gain, maintain, and apply such knowledge*. Another way of learning is through experience. Learning from experience counted as a way developers, hackers, and testers learned about security vulnerabilities (section 3.1.6). Companies should *help developers gain experience with vulnerabilities and investigate security incidents*, for instance, by assigning new developers to investigations of vulnerabilities of the developed software and related security incidents, or motivating developers to learn about others' experiences with vulnerabilities.

Support: Support developers by providing them with adequate tools and human resources. Developers need to have sufficient expertise to use SATs. Also, they face many challenges that require help in using tools. A reason developers use Stack Overflow to find solutions for security-related problems is that they receive quick responses without negative social judgment (section 3.3.2, section 3.3.3). *Developers should be supported when using analysis tools by providing access to experts to ask their questions about the tools or whenever they need guidance in using them. Expert support needs to be easily available, quick, and without any criticism*. Furthermore, considering developers' needs in different projects and development phases, they need different levels of warnings and information in SATs. Organizations can provide resources (section 3.3.2, section 3.2.3). *Organizations should choose analysis tools that support different levels of detail and speed, have customization options, and have user-friendly interfaces or outputs.*

4.4 Avenues for Academic Research

Our literature analysis sheds light on the landscape of developers' challenges and offers the academic community future avenues of research. While there is a long list of potential research, we focus here on the existing need for research that, if investigated, would contribute the most to the state of the art and practice in SSE.

Experience: The role of experience in SSE must be explored further. Research suggests that overconfidence could lead to vulnerabilities (section 3.1.1 & section 3.1.2). More research is needed to understand the link between the amount of developers' work experience and overconfidence, as well as how it can be reduced. More importantly, due to the lack of clarity about the role of experience (section 3.1.5), more investigation of the relationship between different types of experience (e.g., work, tool use, language, etc.) and the ability to engineer secure software is necessary. Also, as some research suggests, security training is not a significant factor in preventing vulnerabilities (section 3.1.5); more investigation is necessary to understand why education is not working and how developers' education can be adjusted or even reformed to increase the positive impact on the security of engineered software.

Learning strategies: More research is needed to determine the most effective learning strategies for SSE. As the demography of developers is shifting toward developers with a wide variety of educational backgrounds [66], we need to know which learning strategies could be more effective for such demographics. To bridge this gap, the academic community needs to consider findings related to: (1) the common practice of using unofficial resources such as Stack Overflow (section 3.3.3), (2) how developers who used official resources were able to develop more secure code than those who used unofficial resources(section 3.3.3), (3) how developers have difficulties in finding a correlation between theorized vulnerabilities that they had learned about in school and the security problem at hand (section 3.1.6). Therefore, more research is needed to understand how teaching "on the spot" needs to be provided.

Code examples in tools: With the use of code examples from diverse sources, more studies are needed to determine how developers use these examples. Code examples appear to play an important role in SSE (section 3.3.1, section 3.3.2). Developers might greatly benefit from usable, effective, and secure code examples in the documentation of APIs and protocols, as well as in the notifications produced by analysis tools in particular. Developers usually use code examples from online sources, and many of those examples could have vulnerabilities (section 3.3.3). Some of the most promising questions to investigate in the context of SSE are: What code examples are useful? How are developers using code examples? And what are developers' challenges with using them?

Focus on demographics: It seems that comparing developers to either end users or students in studies on software security can be counterproductive. One position paper provided a research agenda for developer studies by considering lessons learned from usable security for end users [5]. This is a good starting point for developer-centered studies, but this type of comparison is associated with limitations [66]. To understand these limitations, Gorski et al. [39] drew the attention of API developers to the fact that, from the developers' point of view, "end-user centered security warning guidelines cannot be applied directly" for designing warnings for developers. Applying findings about end-users to the context of developer-centered security could cause researchers to miss some important aspects such as organizational factors (section 3.2), motivations (section 3.1.3), tool availability, and requirements related to tools (section 3.3), etc., resulting in them not fully understanding the situations that developers find themselves in. For this reason, students cannot be used in studies that aim to fully understand professional developers' challenges in SSE, which is one of the major concerns related to developer studies [5, 27, 57–60]. As shown in Fig 1, organizational dimensions (e.g., culture, policies) have significant influence on SSE for professional developers, which is not the case for students.

While some studies have suggested using students instead of professional developers, due to the access difficulties, several findings (section 3.1.5, section 3.2.2, section 3.1.6, section 3.1.3, section 3.3.2) suggest that differences between these two demographics undercut the argument in favor of their similarity as subjects. Finally, we would like to echo the finding of Naiakshina et al. [57] as they discovered that company developers could develop more secure software than students, which could be a result of the company context as a possible key factor. Therefore, further research is needed to examine the differences between students and developers.

ACKNOWLEDGMENTS

Many thanks to Mohammad Tahaei, Artemij Voskobojnikov, Borke Obada, Masoud Mehrabi Koushki, Yue Huang, and Tiffany Quon for their input and feedback. Stylistic and copy editing by Eva van Emden helped to improve readability of this paper. This research has been supported by the Huawei-UBC Software Engineering Technology Research Program.

REFERENCES

[1] Rabe Abdalkareem, Emad Shihab, and Juergen Rilling. 2017. On code reuse from stackoverflow: An exploratory study on android apps. *Information and Software Technology* 88 (2017), 148–158.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Simson L. Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2017. Comparing the Usability of Cryptographic APIs.. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, San Jose, CA, 154–171. http://dblp.uni-trier.de/db/conf/sp/sp2017.html#Acar0FGKMS17

[3] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE Press, San Jose, CA, 289–305. https://doi.org/10.1109/SP.2016.25

[4] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2017. How internet resources might be helping you develop faster but less securely. *IEEE Security & Privacy* 15, 2 (2017), 50–60.

[5] Yasemin Acar, Sascha Fahl, and Michelle L Mazurek. 2016. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, 3–8.

[6] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L. Mazurek, and Sascha Fahl. 2017. Security Developer Studies with GitHub Users: Exploring a Convenience Sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. USENIX Association, Santa Clara, CA, 81–95. https://www.usenix.org/conference/soups2017/technical-sessions/presentation/acar

[7] Debi Ashenden and Darren Lawrence. 2016. Security dialogues: Building better relationships between security and business. *IEEE Security & Privacy* 14, 3 (2016), 82–87.

[8] Hala Assal and Sonia Chiasson. 2018. Motivations and amotivations for software security. In *SOUPS Workshop on Security Information Workers (WSIW). USENIX Association*. Baltimore, MD, USA, 1–4.

[9] Hala Assal and Sonia Chiasson. 2018. Security in the Software Development Lifecycle. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, 281–296. https://www.usenix.org/conference/soups2018/presentation/assal

[10] Hala Assal and Sonia Chiasson. 2019. <i>'Think Secure from the Beginning'</i>: A Survey with Software Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300519

[11] Nathaniel Ayewah and William Pugh. 2008. A Report on a Survey and Study of Static Analysis Users. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems* (Seattle, Washington) *(DEFECTS '08)*. Association for Computing Machinery, New York, NY, USA, 1–5. https://doi.org/10.1145/1390817.1390819

[12] Nathaniel Ayewah and William Pugh. 2010. The Google FindBugs Fixit. In *Proceedings of the 19th International Symposium on Software Testing and Analysis* (Trento, Italy) *(ISSTA '10)*. Association for Computing Machinery, New York, NY, USA, 241–252. https://doi.org/10.1145/1831708.1831738

[13] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix. 2008. Using Static Analysis to Find Bugs. *IEEE Software* 25, 5 (Sep. 2008), 22–29.

https://doi.org/10.1109/MS.2008.130
[14] Dejan Baca and Bengt Carlsson. 2011. Agile Development with Security Engineering Activities. In *Proceedings of the 2011 International Conference on Software and Systems Process* (Waikiki, Honolulu, HI, USA) *(ICSSP '11)*. Association for Computing Machinery, New York, NY, USA, 149–158. https://doi.org/10.1145/1987875.1987900
[15] Dejan Baca, Kai Petersen, Bengt Carlsson, and Lars Lundberg. 2009. Static code analysis to detect software security vulnerabilities-does experience matter?. In *2009 International Conference on Availability, Reliability and Security*. IEEE, IEEE Press, Fukuoka, Japan, 804–810.
[16] Wei Bai, Omer Akgul, and Michelle L. Mazurek. 2019. A Qualitative Investigation of Insecure Code Propagation from Online Forums. In *2019 IEEE Cybersecurity Development (SecDev)*. IEEE Press, McLean, VA, 34–48. https://doi.org/10.1109/SecDev.2019.00016
[17] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason I Hong, and Lorrie Faith Cranor. 2014. The privacy and security behaviors of smartphone app developers. In *Proc. of Workshop on Usable Security*. Internet Society, San Diego, CA, USA, 1–10.
[18] Steffen Bartsch. 2011. Practitioners' Perspectives on Security in Agile Development. In *Proceedings of the 2011 Sixth International Conference on Availability, Reliability and Security (ARES '11)*. IEEE Computer Society, USA, 479–484. https://doi.org/10.1109/ARES.2011.82
[19] Jason Bau, Frank Wang, Elie Bursztein, Patrick Mutchler, and John C Mitchell. 2012. Vulnerability factors in new web applications: Audit tools, developer selection & languages. *Stanford, Tech. Rep* (2012).
[20] Deanna D. Caputo, Shari Lawrence Pfleeger, M. Angela Sasse, Paul Ammann, Jeff Offutt, and Lin Deng. 2016. Barriers to Usable Security? Three Organizational Case Studies. *IEEE Security Privacy* 14, 5 (2016), 22–32. https://doi.org/10.1109/MSP.2016.95
[21] George Chalhoub, Ivan Flechais, Norbert Nthala, and Ruba Abu-Salma. 2020. Innovation Inaction or In Action? The Role of User Experience in the Security and Privacy Design of Smart Home Cameras. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX Association, Online, 185–204. https://www.usenix.org/conference/soups2020/presentation/chalhoub
[22] Mengsu Chen, Felix Fischer, Na Meng, Xiaoyin Wang, and Jens Grossklags. 2019. How Reliable is the Crowdsourced Knowledge of Security Implementation?. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 536–547. https://doi.org/10.1109/ICSE.2019.00065
[23] Sen Chen, Ting Su, Lingling Fan, Guozhu Meng, Minhui Xue, Yang Liu, and Lihua Xu. 2018. Are Mobile Banking Apps Secure? What Can Be Improved?. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 797–802. https://doi.org/10.1145/3236024.3275523
[24] Maria Christakis and Christian Bird. 2016. What Developers Want and Need from Program Analysis: An Empirical Study. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE 2016)*. Association for Computing Machinery, New York, NY, USA, 332–343. https://doi.org/10.1145/2970276.2970347
[25] Symantec Corporation. [n.d.]. Symantec Internet Security Threat Report. https://docs.broadcom.com/doc/istr-22-2017-en, publisher=Broadcom Inc. Accessed: 2021-05-03.
[26] Catherine Courage and Kathy Baxter. 2005. Appendix F - Affinity Diagram. In *Understanding Your Users: A Practical Guide to User Requirements Methods, Tools, and Techniques* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[27] Anastasia Danilova, Alena Naiakshina, Johanna Deuter, and Matthew Smith. 2020. Replication: On the Ecological Validity of Online Security Developer Studies: Exploring Deception in a Password-Storage Study with Freelancers. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX Association, Online, 165–183. https://www.usenix.org/conference/soups2020/presentation/danilova
[28] Anastasia Danilova, Alena Naiakshina, and Matthew Smith. 2020. One Size Does Not Fit All: A Grounded Theory and Online Survey Study of Developer Preferences for Security Warning Types. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 136–148. https://doi.org/10.1145/3377811.3380387
[29] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. 2017. Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2187–2200. https://doi.org/10.1145/3133956.3134059
[30] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling Static Analyses at Facebook. *Commun. ACM* 62, 8 (July 2019), 62–70. https://doi.org/10.1145/3338112
[31] Anne Edmundson, Brian Holtkamp, Emanuel Rivera, Matthew Finifter, Adrian Mettler, and David Wagner. 2013. An Empirical Study on the Effectiveness of Security Code Review. In *Proceedings of the 5th International Conference on Engineering Secure Software and Systems* (Paris, France) *(ESSoS'13)*. Springer-Verlag, Berlin, Heidelberg, 197–212. https://doi.org/10.1007/978-3-642-36563-8_14
[32] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An Empirical Study of Cryptographic Misuse in Android Applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 73–84. https://doi.org/10.1145/2508859.2516693
[33] Sascha Fahl, Marian Harbach, Henning Perl, Markus Koetter, and Matthew Smith. 2013. Rethinking SSL Development in an Appified World. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 49–60. https://doi.org/10.1145/2508859.2516655
[34] Martha S. Feldman and Brian T. Pentland. 2003. Reconceptualizing Organizational Routines as a Source of Flexibility and Change. *Administrative Science Quarterly* 48, 1 (2003), 94–118. http://www.jstor.org/stable/3556620
[35] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. 2017. Stack overflow considered harmful? the impact of copy&paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE Press, San Jose, CA, 121–136.
[36] S. Frey, A. Rashid, P. Anthonysamy, M. Pinto-Albuquerque, and S. A. Naqvi. 2019. The Good, the Bad and the Ugly: A Study of Security Decisions in a Cyber-Physical Systems Game. *IEEE Transactions on Software Engineering* 45, 5 (May 2019), 521–536. https://doi.org/10.1109/TSE.2017.2782813
[37] Joseph Giacomin. 2014. What is human centred design? *The Design Journal* 17, 4 (2014), 606–623.
[38] Peter Leo Gorski. 2021. Information flows to support software developers in using security APIs. (2021).
[39] Peter Leo Gorski, Yasemin Acar, Luigi Lo Iacono, and Sascha Fahl. 2020. Listen to Developers! A Participatory Design Study on Security Warnings for Cryptographic APIs. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376142
[40] M. Green and M. Smith. 2016. Developers are Not the Enemy!: The Need for Usable Security APIs. *IEEE Security Privacy* 14, 5 (Sep. 2016), 40–46. https://doi.org/10.1109/MSP.2016.111
[41] Peter Hamm, David Harborth, and Sebastian Pape. 2019. A Systematic Analysis of User Evaluations in Security Research. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*. 1–7.
[42] Julie M. Haney, Mary Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. 2018. "We make it a big deal in the company": Security Mindsets in Organizations that Develop Cryptographic Products. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, 357–373. https://www.usenix.org/conference/soups2018/presentation/haney-mindsets
[43] Nasif Imtiaz, Akond Rahman, Effat Farhana, and Laurie Williams. 2019. Challenges with Responding to Static Analysis Tool Alerts. In *Proceedings of the 16th International Conference on Mining Software Repositories* (Montréal, QC, Canada) *(MSR '19)*. IEEE Press, Montreal, Quebec, Canada, 245–249. https://doi.org/10.1109/MSR.2019.00049
[44] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. Association for Computing Machinery, New York, NY, USA, 672–681.
[45] Timothy P. Johnson. 2014. Snowball Sampling: Introduction. (2014). https://doi.org/10.1002/9781118445112.stat05720 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118445112.stat05720
[46] Tamara Lopez, Helen Sharp, Thein Tun, Arosha Bandara, Mark Levine, and Bashar Nuseibeh. 2019. Talking about Security with Professional Developers. In *Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice (CESSER-IP '19)*. IEEE Press, Montreal, Quebec, Canada, 34–40. https://doi.org/10.1109/CESSER-IP.2019.00014
[47] Tamara Lopez, Helen Sharp, Thein Tun, Arosha K. Bandara, Mark Levine, and Bashar Nuseibeh. 2019. "Hopefully We Are Mostly Secure": Views on Secure Code in Professional Practice. In *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '19)*. IEEE Press, Montreal, Quebec, Canada, 61–68. https://doi.org/10.1109/CHASE.2019.00023
[48] Tamara Lopez, Thein Tun, Arosha Bandara, Mark Levine, Bashar Nuseibeh, and Helen Sharp. 2019. An Anatomy of Security Conversations in Stack Overflow. In *Proceedings of the 41st International Conference on Software Engineering:*

Software Engineering in Society (ICSE-SEIS '19). IEEE Press, Montreal, Quebec, Canada, 31–40. https://doi.org/10.1109/ICSE-SEIS.2019.00012
[49] Check Point Software Technologies Ltd. [n.d.]. Cyber Security Report 2020. https://www.ntsc.org/assets/pdfs/cyber-security-report-2020.pdf, publisher=CHECK POINT. Accessed: 2021-05-03.
[50] Na Meng, Stefan Nagy, Danfeng (Daphne) Yao, Wenjie Zhuang, and Gustavo Arango Argoty. 2018. Secure Coding Practices in Java: Challenges and Vulnerabilities. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 372–383. https://doi.org/10.1145/3180155.3180201
[51] merriam webster. [n.d.]. Confidence. https://www.merriam-webster.com/dictionary/confidence, publisher=Merriam-Webster. Accessed: 2021-07-20.
[52] merriam webster. [n.d.]. Deterrents. https://www.merriam-webster.com/dictionary/deterrent, publisher=Merriam-Webster. Accessed: 2021-07-20.
[53] merriam webster. [n.d.]. Mindset. https://www.merriam-webster.com/dictionary/mindset, publisher=Merriam-Webster. Accessed: 2019-10-03.
[54] Mohammad Tahaei and Kami Vaniea. 2019. A Survey on Developer-Centred Security. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. 129–138. https://doi.org/10.1109/EuroSPW.2019.00021
[55] Azadeh Mokhberi, Tiffany Quon, and Konstantin Beznosov. 2021. What makes security-related code examples different. In *SOUPS Workshop on Security Information Workers (WSIW). USENIX Association*.
[56] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. 2016. Jumping through Hoops: Why Do Java Developers Struggle with Cryptography APIs?. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 935–946. https://doi.org/10.1145/2884781.2884790
[57] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, and Matthew Smith. 2020. On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376791
[58] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. 2019. "If You Want, I Can Store the Encrypted Password": A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article 140, 12 pages. https://doi.org/10.1145/3290605.3300370
[59] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 311–328. https://doi.org/10.1145/3133956.3134082
[60] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, and Matthew Smith. 2018. Deception Task Design in Developer Password Studies: Exploring a Student Sample. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, 297–313. https://www.usenix.org/conference/soups2018/presentation/naiakshina
[61] Chitu Okoli. 2015. A guide to conducting a standalone systematic literature review. *Communications of the Association for Information Systems* 37, 1 (2015), 43.
[62] Daniela Oliveira, Marissa Rosenthal, Nicole Morin, Kuo-Chuan Yeh, Justin Cappos, and Yanyan Zhuang. 2014. It's the Psychology Stupid: How Heuristics Explain Software Vulnerabilities and How Priming Can Illuminate Developer's Blind Spots. In *Proceedings of the 30th Annual Computer Security Applications Conference* (New Orleans, Louisiana, USA) *(ACSAC '14)*. Association for Computing Machinery, New York, NY, USA, 296–305. https://doi.org/10.1145/2664243.2664254
[63] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A. DeLong, Justin Cappos, and Yuriy Brun. 2018. API Blindspots: Why Experienced Developers Write Vulnerable Code. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, 315–328. https://www.usenix.org/conference/soups2018/presentation/oliveira
[64] Stack Overflow. [n.d.]. Stack Overflow. https://stackoverflow.com/. Accessed: 2019-12-03.
[65] Hernan Palombo, Armin Ziaie Tabari, Daniel Lende, Jay Ligatti, and Xinming Ou. 2020. An Ethnographic Understanding of Software (In)Security and a Co-Creation Model to Improve Secure Software Development. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX Association, Online, 205–220. https://www.usenix.org/conference/soups2020/presentation/palombo
[66] Olgierd Pieczul, Simon Foley, and Mary Ellen Zurko. 2017. Developer-Centered Security and the Symmetry of Ignorance. In *Proceedings of the 2017 New Security Paradigms Workshop* (Santa Cruz, CA, USA) *(NSPW 2017)*. Association for Computing Machinery, New York, NY, USA, 46–56. https://doi.org/10.1145/3171533.3171539
[67] Andreas Poller, Laura Kocksch, Sven Türpe, Felix Anand Epp, and Katharina Kinder-Kurlanda. 2017. Can Security Become a Routine? A Study of Organizational Change in an Agile Software Development Group. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing* (Portland, Oregon, USA) *(CSCW '17)*. Association for Computing Machinery, New York, NY, USA, 2489–2503. https://doi.org/10.1145/2998181.2998191
[68] Lutz Prechelt. 2010. Plat_Forms: A web development platform comparison by an exploratory experiment searching for emergent platform properties. *IEEE Transactions on Software Engineering* 37, 1 (2010), 95–108.
[69] Akond Rahman, Asif Partho, Patrick Morrison, and Laurie Williams. 2018. What Questions Do Programmers Ask about Configuration as Code?. In *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering* (Gothenburg, Sweden) *(RCoSE '18)*. Association for Computing Machinery, New York, NY, USA, 16–22. https://doi.org/10.1145/3194760.3194769
[70] Andrew Ruef, Michael Hicks, James Parker, Dave Levin, Michelle L. Mazurek, and Piotr Mardziel. 2016. Build It, Break It, Fix It: Contesting Secure Development. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 690–703. https://doi.org/10.1145/2976749.2978382
[71] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. 2018. Lessons from Building Static Analysis Tools at Google. *Commun. ACM* 61, 4 (March 2018), 58–66. https://doi.org/10.1145/3188720
[72] Caitlin Sadowski, Jeffrey van Gogh, Ciera Jaspan, Emma Soderberg, and Collin Winter. 2015. Tricorder: Building a Program Analysis Ecosystem. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, Florence, Italy, 598–608. https://doi.org/10.1109/ICSE.2015.76
[73] Pasquale Salza, Fabio Palomba, Dario Di Nucci, Andrea De Lucia, and Filomena Ferrucci. 2020. Third-party libraries in mobile apps. *Empirical Software Engineering* 25, 3 (2020), 2341–2377.
[74] Elizabeth B-N Sanders and Pieter Jan Stappers. 2008. Co-creation and the new landscapes of design. *Co-design* 4, 1 (2008), 5–18.
[75] Jesper Simonsen, Connie Svabo, Sara Malou Strandvad, Kristine Samson, Morten Hertzum, and Ole Erik Hansen. 2014. *Situated Design Methods*. The MIT Press.
[76] Justin Smith, Lisa Nguyen Quang Do, and Emerson Murphy-Hill. 2020. 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)*. USENIX Association, Online, 221–238. https://www.usenix.org/conference/soups2020/presentation/smith
[77] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. 2019. How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool. *IEEE Transactions on Software Engineering* 45, 9, 877–897. https://doi.org/10.1109/TSE.2018.2810116
[78] Mohammad Tahaei, Kami Vaniea, Konstantin (Kosta) Beznosov, and Maria K Wolters. 2021. *Security Notifications in Static Analysis Tools: Developers' Attitudes, Comprehension, and Ability to Act on Them*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3411764.3445616
[79] Tyler W. Thomas, Heather Lipford, Bill Chu, Justin Smith, and Emerson Murphy-Hill. 2016. What Questions Remain? An Examination of How Developers Understand an Interactive Static Analysis Tool. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, Denver, CO. https://www.usenix.org/conference/soups2016/workshop-program/wsiw16/presentation/thomas
[80] Tyler W. Thomas, Madiha Tabassum, Bill Chu, and Heather Lipford. 2018. Security During Application Development: An Application Security Expert Perspective. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173836
[81] Mieke Van der Bijl-Brouwer and Kees Dorst. 2017. Advancing the strategic impact of human-centred design. *Design Studies* 53 (2017), 1–23.
[82] veracode. [n.d.]. STATE OF SOFTWARE SECURITY. https://www.veracode.com/sites/default/files/Resources/Reports/state-of-software-security-volume-7-veracode-report.pdf, publisher=Veracode. Accessed: 2021-05-03.
[83] Jan Vom Brocke, Alexander Simons, Kai Riemer, Bjoern Niehaves, Ralf Plattfaut, and Anne Cleven. 2015. Standing on the shoulders of giants: Challenges and recommendations of literature search in information systems research. *Communications of the association for information systems* 37, 1 (2015), 9.
[84] Daniel Votipka, Desiree Abrokwa, and Michelle L. Mazurek. 2020. Building and Validating a Scale for Secure Software Development Self-Efficacy. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–20. https://doi.org/10.1145/3313831.3376754
[85] Daniel Votipka, Kelsey R. Fulton, James Parker, Matthew Hou, Michelle L. Mazurek, and Michael Hicks. 2020. Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It. In *29th USENIX*

Security Symposium (USENIX Security 20). USENIX Association, Online, 109–126. https://www.usenix.org/conference/usenixsecurity20/presentation/votipka-understanding

[86] Daniel Votipka, Rock Stevens, Elissa Redmiles, Jeremy Hu, and Michelle Mazurek. 2018. Hackers vs. testers: A comparison of software vulnerability discovery processes. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE Press, San Francisco, CA, 374–391.

[87] Charles Weir, Ingolf Becker, and Lynne Blair. 2021. A Passion for Security: Intervening to Help Software Developers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 21–30. https://doi.org/10.1109/ICSE-SEIP52600.2021.00011

[88] Charles Weir, Ingolf Becker, James Noble, Lynne Blair, M. Angela Sasse, and Awais Rashid. 2019. Interventions for Software Security: Creating a Lightweight Program of Assurance Techniques for Developers. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19)*. IEEE Press, Montreal, Quebec, Canada, 41–50. https://doi.org/10.1109/ICSE-SEIP.2019.00013

[89] Charles Weir, Ben Hermann, and Sascha Fahl. 2020. From Needs to Actions to Secure Apps? The Effect of Requirements and Developer Practices on App Security. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 289–305. https://www.usenix.org/conference/usenixsecurity20/presentation/weir

[90] Charles Weir, James Noble, and Awais Rashid. 2020. Challenging Software Developers: Dialectic as a Foundation for Security Assurance Techniques. *Journal of Cybersecurity* (2020), 30. https://doi.org/10.1093/cybsec/tyaa007

[91] Charles Weir, Awais Rashid, and James Noble. 2016. Reaching the Masses: A New Subdiscipline of App Programmer Education. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA) *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 936–939. https://doi.org/10.1145/2950290.2983981

[92] Charles Weir, Awais Rashid, and James Noble. 2017. I'd Like to Have an Argument, Please: Using Dialectic for Effective App Security. (2017). https://doi.org/10.14722/eurousec.2017.23002

[93] Dominik Wermke, Nicolas Huaman, Yasemin Acar, Bradley Reaves, Patrick Traynor, and Sascha Fahl. 2018. A Large Scale Investigation of Obfuscation Use in Google Play. In *Proceedings of the 34th Annual Computer Security Applications Conference* (San Juan, PR, USA) *(ACSAC '18)*. Association for Computing Machinery, New York, NY, USA, 222–235. https://doi.org/10.1145/3274694.3274726

[94] Michael Whitney, Heather Lipford-Richter, Bill Chu, and Jun Zhu. 2015. Embedding Secure Coding Instruction into the IDE: A Field Study in an Advanced CS Course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 60–65. https://doi.org/10.1145/2676723.2677280

[95] Chamila Wijayarathna and Nalin A. G. Arachchilage. 2018. Why Johnny Can't Store Passwords Securely? A Usability Evaluation of Bouncycastle Password Hashing. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (Christchurch, New Zealand) *(EASE'18)*. Association for Computing Machinery, New York, NY, USA, 205–210. https://doi.org/10.1145/3210459.3210483

[96] Jim Witschey, Shundan Xiao, and Emerson Murphy-Hill. 2014. Technical and Personal Factors Influencing Developers' Adoption of Security Tools. In *Proceedings of the 2014 ACM Workshop on Security Information Workers* (Scottsdale, Arizona, USA) *(SIW '14)*. Association for Computing Machinery, New York, NY, USA, 23–26. https://doi.org/10.1145/2663887.2663898

[97] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. 2015. Quantifying Developers' Adoption of Security Tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 260–271. https://doi.org/10.1145/2786805.2786816

[98] Joost F Wolfswinkel, Elfi Furtmueller, and Celeste PM Wilderom. 2013. Using grounded theory as a method for rigorously reviewing literature. *European journal of information systems* 22, 1 (2013), 45–55.

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

[100] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing* (Baltimore, Maryland, USA) *(CSCW '14)*. Association for Computing Machinery, New York, NY, USA, 1095–1106. https://doi.org/10.1145/2531602.2531722

[101] Jing Xie, Heather Lipford, and Bei-Tseng Chu. 2012. Evaluating Interactive Support for Secure Programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2707–2716. https://doi.org/10.1145/2207676.2208665

[102] J. Xie, H. R. Lipford, and B. Chu. 2011. Why do programmers make security errors?. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE Press, 161–164. https://doi.org/10.1109/VLHCC.2011.6070393

[103] Mohammad Zarour, Mamdouh Alenezi, and Khalid Alsarayrah. 2020. Software Security Specifications and Design: How Software Engineers and Practitioners Are Mixing Things Up. In *Proceedings of the Evaluation and Assessment in Software Engineering*. Association for Computing Machinery, New York, NY, USA, 451–456. https://doi.org/10.1145/3383219.3383284

[104] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. Are Code Examples on an Online Q&amp;A Forum Reliable? A Study of API Misuse on Stack Overflow. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 886–896. https://doi.org/10.1145/3180155.3180260

Appendices

5 FRAMEWORK

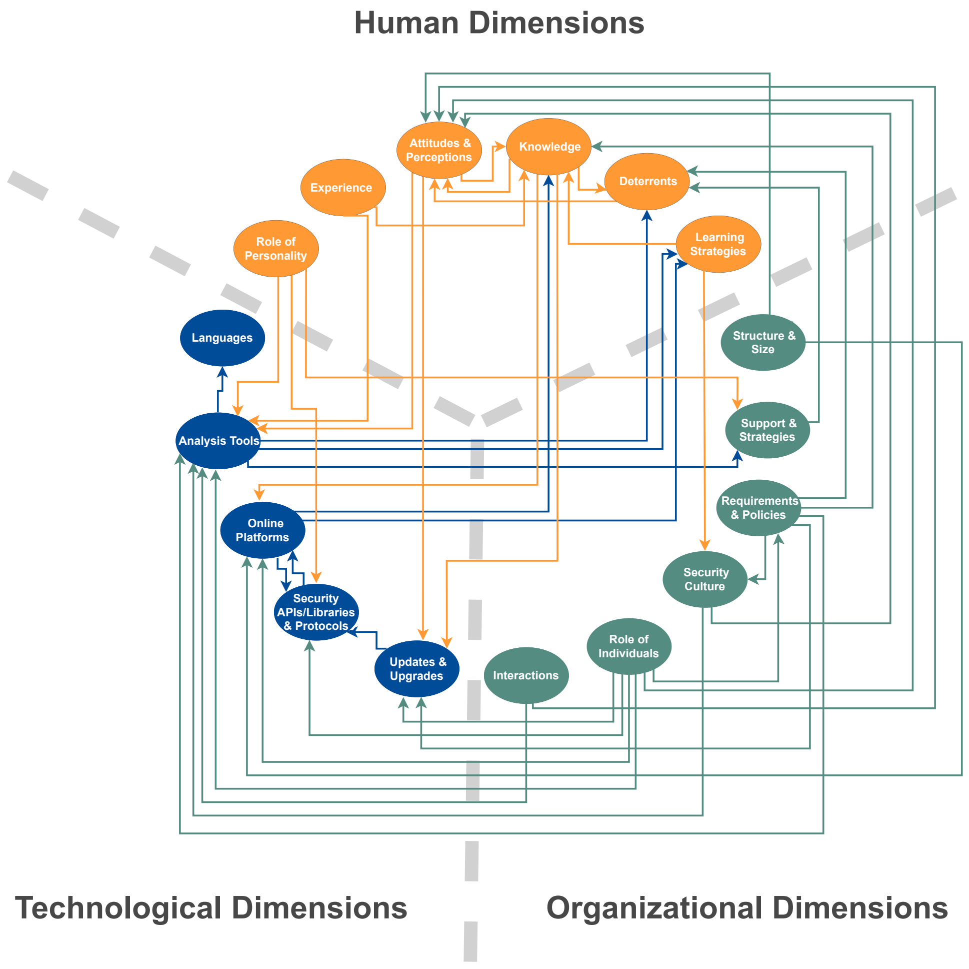

Figure 1: The framework of developers' challenges and their interplay. In this framework, ovals are identified areas of challenge (human dimensions: orange,organizational dimensions: blue, and technological dimensions: green). Arrows indicate correlations between the areas of challenge. These correlations can be one or more of the following: (1) challenges are linked, (2) one negatively affects the other, (3) one positively affects the other, (4) one causes new challenges in the other area. For more details about the interactions showed by an arrow in the framework, see the related challenge descriptions.

6 FRAMEWORK2

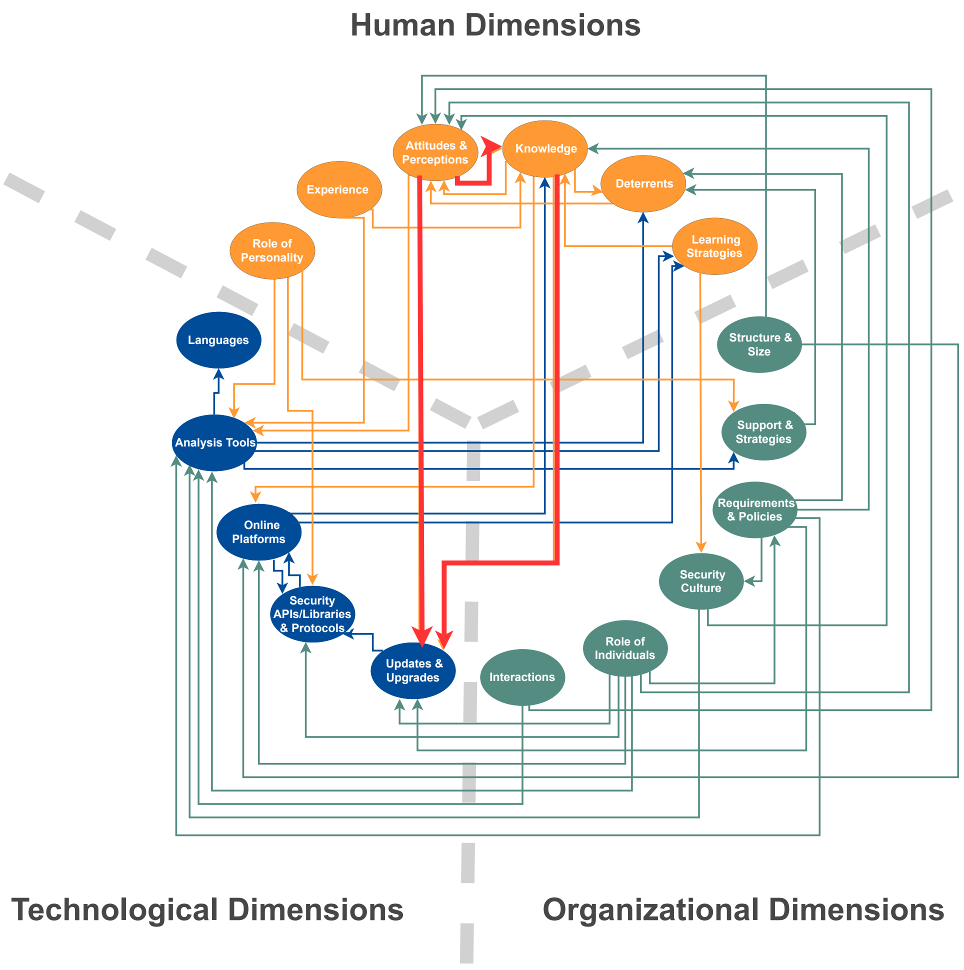

Figure 2: The loop of challenges can be seen in red color

7 SEARCH STRATEGY

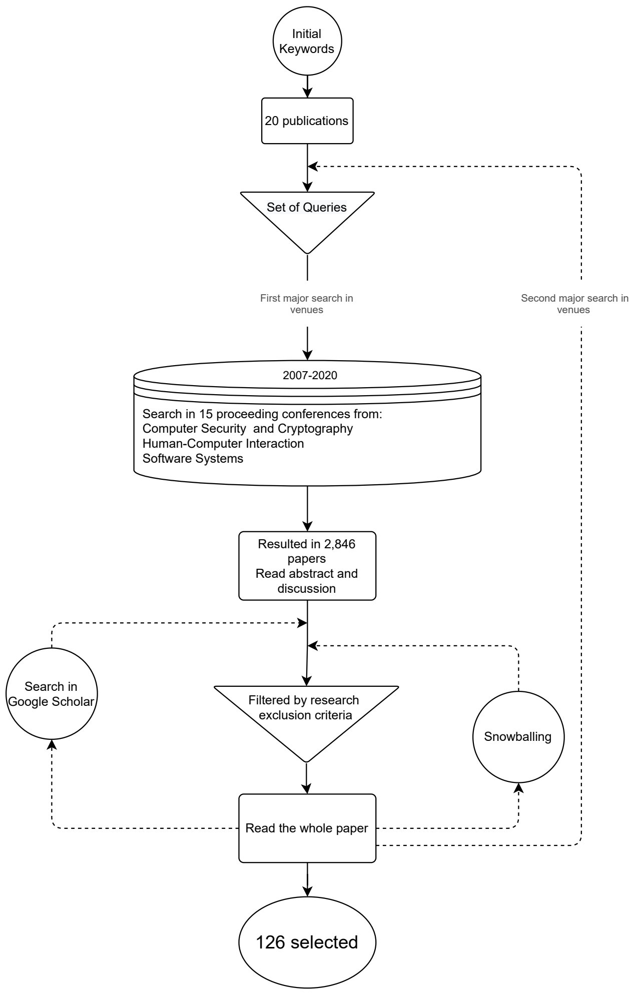
Initial Keywords
20 publications
Set of Queries
First major search in venues
Second major search in venues
2007-2020
Search in 15 proceeding conferences from:
Computer Security and Cryptography
Human-Computer Interaction
Software Systems
Resulted in 2,846 papers
Read abstract and discussion
Search in Google Scholar
Filtered by research exclusion criteria
Snowballing
Read the whole paper
126 selected

Figure 3: Search strategy.

8 SUMMARY OF REVIEWED PUBLICATIONS

Reference No.	Study Design	Human						Organizational						Technological				
	Method-Type of Participants-Number of Participants	Role of Personality	Attitudes & Per.	Deterrents	Knowledge	Experience	Learning Strategies	Security Culture	Requir. & Policies	Support and Strategies	Structure & Size	Role of Individuals in Org.	Interactions	Sec. APIs/Libs., Protocols	Analyses Tools	Online Platforms	Updates & Upgrades	Languages
Σ		07	18	08	14	14	06	09	14	06	07	08	06	08	17	13	03	05
2020—- Chalhoub [21]	Interview-UX designers, developers, ...-20									■			■					
———Danilova [27]	online study-Freelance developers-43					■			■									■
———Danilova [28]	Interview-dev, students-14,12+focus group 7 researchers+survey,50 dev		■						■		■				■			
———Naiakshina [57]	Online lab study-Students, Freelancers, & developers-36						■		■									
———Palombo [65]	1,5 years of observationl study (ethnographic study)		■									■	■					
———Smith [76]	Walkthrough, user study-developers-12														■			
———Votipka [84]	Survey-developers-311	■				■												
———Votipka [85]	94submissions of four BIBIFI contests				■	■												
———Weir [89]	Survey-developers-335								■			■						
———Weir [90]	Interview-app sec. experts-12		■		■								■			■	■	
2019—Assal [10]	Survey-developers-123		■	■		■				■	■							
———Bai [16]	survey,interview-GitHub developers-133,15	■	■		■													
———Chen [22]	Analysis and evaluation of Stack Overflow posts															■		
———Frey [36]	lab study-sec. experts, computer scientists, managers-43	■																
———Imtiaz [43]	Analysis of 280 Stack Overflow questions														■			
———Lopez [47]	Interview-managers, developers, tester-7		■															
———Lopez [46]	Ethnographic studies (sets of workshops-developers-70+							■										
———Lopez [48]	Study on Stack Overflow		■													■		
———Naiakshina [58]	Lab study-freelancer developers-40				■	■			■									■
———Salza [73]	Survey-developers-73																■	
———Weir [88]	Interview, workshops -sec. experts, developers-15,3 teams							■										
2018—Assal [8]	Semi structured interview-developers-13			■	■													
———Assal [9]	Interview-developers-13		■	■	■			■	■	■	■			■				
———Chen [23]	Interview-bank entities-7, survey-developers-20 noinfo		■		■													
———Haney [42]	Semi structured interview-mix-21						■	■		■							■	
———Meng [50]	Empirical study on Stack Overflow posts				■									■		■		
———Oliveira [63]	Online task study-developers, students-70,39	■				■												
———Rahman [69]	Study on Stack Overflow															■		
———Smith [77]	Lab study-Students, developers-5,5														■			
———Thomas [80]	Semi structured interview-sec. experts-32		■	■						■	■	■		■				
———Votipka [86]	Interview-testers,hackers-25	■											■					
———Wermke [93]	Survey-developers-308				■													
———Wijayarathna [95]	Lab study-developers-10				■									■		■		
———Zhang [104]	Analysis of 217,818 Stack Overflow posts															■		
2017—Acar [2]	Controlled experiment study-developers-256													■				
———Acar [4]	Survey-developers-295		■			■												
———Acar [6]	Online experimental study-developers-307		■			■												
———Abdalkareem [1]	Analysis and evaluation of 22 open source Android apps															■		
———Derr [29]	Survey-developers-203																■	
———Fischer [35]	Study on Stack Overflow															■		
———Poller [67]	survey, survey, observation, interview-mix-15,12,23,15		■	■				■	■		■	■						
———Weir [92]	Interview-sec. experts-12											■						
———Naiakshina [59]	Lab study-students-20		■		■		■		■									
2016—Acar [3]	Survey, lab experiment-developers-295,54															■		
———Ashenden [7]	Three pilot workshops-developers-18							■										
———Caputo [20]	Three organizations studied							■										
———Christakis [24]	Interview, Survey-developers-5, 375				■										■			
———Nadi [56]	survey, survey-developers-11,37						■							■		■		
———Ruef [70]	Evaluation of work by 116 teams of three BIBIFI contests																	■
———Thomas [79]	Observation behaviour-developers-13 noinfo														■			
2015—Sadowski [72]	In-situ evaluation- Google developers														■			
———Whitney [94]	Two field study-students-72		■															
———Witschey [97]	Survey, survey-developers-14, 61 noinfo					■		■	■			■	■		■			
2014—Balebako [17]	Semi structured interview, Survey-developers, mix-13, 228				■		■				■					■		
———Oliveira [62]	Survey-developers-47	■	■				■											
———Witschey [96]	Interview-developers-42	■		■		■									■			
———Xiao [100]	Semi structured interview-developers-42		■		■	■			■	■		■	■		■			
2013—Edmundson [31]	Online tasks-developers-30					■												
———Egele [32]	Empirical Study													■				
———Fahl [33]	Interview-developers-14													■				
———Johnson [44]	Lab study-developers-20			■											■			
2012—Xie [101]	Lab study-students, developers-18, 9								■						■			
———Bau [19]	Lab study-Sturtup developers, freelancers-19, 8																	■
2011—Baca [14]	Interview-developers-12					■									■			
———Bartsch [18]	Semi structured interview-majority developer-10								■									
———Xie [102]	Semi structured interview-developers-15		■	■				■	■		■							
2010—Ayewah [12]	User study-developers-700+														■			
———Prechelt [68]	3 field study-developers-27																	■
2009—Baca [15]	Lab study-developers-34					■									■			
2008—Ayewah [13]	Survey,interview-developers-400+, 12								■						■			
———Ayewah [11]	Lab study-students-12														■			
2007—Woon [99]	Survey-systems professionals (includes developers)-184											■						

Table 1: The list of all reviewed publications grouped by theme (we were referring to professional developers by the term developers or dev). In this table, the greyed cells indicate that those papers have been cited on that specific area of challenge.