"Think secure from the beginning": A Survey with Software Developers

Hala Assal
School of Computer Science
Carleton University
Ottawa, ON, Canada
HalaAssal@scs.carleton.ca

Sonia Chiasson
School of Computer Science
Carleton University
Ottawa, ON, Canada
Chiasson@scs.carleton.ca

ABSTRACT

Vulnerabilities persist despite existing software security initiatives and best practices. This paper focuses on the human factors of software security, including human behaviour and motivation. We conducted an online survey to explore the interplay between developers and software security processes, *e.g.*, we looked into how developers influence and are influenced by these processes. Our data included responses from 123 software developers currently employed in North America who work on various types of software applications.

Whereas developers are often held responsible for security vulnerabilities, our analysis shows that the real issues frequently stem from a lack of organizational or process support to handle security throughout development tasks. Our participants are self-motivated towards software security, and the majority did not dismiss it but identified obstacles to achieving secure code. Our work highlights the need to look beyond the individual, and take a holistic approach to investigate organizational issues influencing software security.

CCS CONCEPTS

• **Security and privacy → Software and application security**; **Human and societal aspects of security and privacy**.

KEYWORDS

Security, Survey, HCI for development, Secure programming

CHI 2019, May 4–9, 2019, Glasgow, Scotland Uk

ACM ISBN 978-1-4503-5970-2/19/05…$15.00
https://doi.org/10.1145/3290605.3300519

ACM Reference Format:
Hala Assal and Sonia Chiasson. 2019. *"Think secure from the beginning"*: A Survey with Software Developers. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland Uk.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3290605.3300519

1 INTRODUCTION

Software security focuses on the resistance of applications to vulnerabilities exercised through malicious exploitations or unintentional triggers [2]. Best practices and initiatives have been proposed to promote the inclusion of security throughout the Software Development Lifecycle (SDLC) (*e.g.*, [19, 41, 48, 55]) in part to address such vulnerabilities. However, vulnerabilities persist, impact millions of users [22], and extend beyond traditional computing systems [31, 50, 51].

Developers are often blamed for vulnerabilities [6] and are sometimes viewed as the "weakest link" who just need to do more [28]. However, recent user-centric research has focused on software developers as users who critically need support when dealing with the implementation of software that adequately addresses security [6, 28, 49].

In this paper, we take a human-centric approach to address an under-investigated research area—the interplay between the developer and the process of managing software security. We focus on understanding how the human actors (*e.g.*, developers) deal with, and influence, this process. Although we do not focus on technologies to support secure software development, this work can help inform the design of these technologies. We note that security vulnerabilities could be unintentional or could be introduced to a system out of malice. In this paper, we focus on supporting developers avoid unintentional vulnerabilities; malicious developers are thus out of the scope of this work. In particular, this paper addresses the following three research questions. ***RQ1:*** *How does security fit in the development lifecycle in real life?* ***RQ2:*** *What are the current motivators and deterrents to developers paying attention to security?* ***RQ3:*** *Does the development methodology, company size, or adopting Test-Driven Development (TDD) influence software security?*

To answer these questions, we conducted an online survey study with a representative sample of 123 software developers from North America. The survey focuses on how developers and their teams direct their efforts towards software security, as well as strategies developers employ to deal with security. We also explore developers' work motivation styles, their motivation towards software security, as well as factors that may deter developers from addressing security.

Our study shows that efforts towards software security vary; in extreme cases, security is consistently disregarded throughout the SDLC but most participants reported at least some attention to it. Development methodology had no significant effects on our results. The use of TDD was most influential, while company size had moderate influence.

In general, our results are promising for software security and suggest that developers do not intentionally disregard it. Our participants have a good understanding of software security and generally oppose statements that imply ignoring or deferring security, even though it is typically not their primary objective [14, 28, 47]. However, our analysis identified systemic barriers to achieving secure code, *e.g.*, the lack of a security plan. This highlights the need to investigate and address organizational issues that lead to insecure practices.

2 RELATED WORK

In their overview of the usable security field, Garfinkel and Lipford [25] highlight the shortage of human factors security research that focuses on software developers. Naiakshina *et al.* [42] cautioned that researchers do not have the same expertise in studies with developers as with typical end-users, and they discussed how different study designs can help investigate different research questions. Pieczul *et al.* [49] discussed challenges facing usable security research for developers and highlighted the need for deeper understanding of the continuously evolving field of software development. We now discuss recent research on this subject.

Developers' Abilities and Expertise. Developers and their lack of security education are frequently cited as the reason for vulnerabilities [47]. The assumption is that if developers learned about security, they could avoid vulnerabilities [11, 69]. Some argue the reason might be because security guidelines do not exist or are not mandated by the companies [67, 70, 73], or that developers might lack the ability [47] or proper expertise [13] to identify vulnerabilities.

Baca *et al.* [13] found that developers' general software development experience did not have the expected positive impact on the correctness of identifying vulnerabilities. Oliveira *et al.* [47] argued that developers and security education are not the root causes of security vulnerabilities. They explained that throughout their tasks, developers are consumed with solving problems that assume common cases, whereas vulnerabilities are usually unexpected corner cases [47] that are cognitively-demanding to identify [56].

Security Tools and Methodologies. Approaches for improved code security include advocating for the use of Static-code Analysis Tools (SATs) [17, 33, 35], reducing their false positives [45, 63], and using innovative methods to assist in vulnerability discovery [29, 63, 74]. However, despite SATs' benefits [10, 20], they are not widely used [10, 34].

Security tool adoption: The company's policies and its overall security culture are among the main factors for encouraging developers to adopt new security tools [70, 73]. Developers' positive perception of the usefulness of security to their applications also encourages security tool adoption [68], whereas tools' complexity discourages it [67, 70].

Improving tools' usability: Smith *et al.* [56] proposed an approach for building tools that support developers' information needs while analyzing vulnerabilities. They identified 17 categories of information that developers seek during the analysis of SAT warnings [56, 57]. These included questions regarding understanding vulnerabilities, attacks that might exploit these vulnerabilities, alternative fixes, and whether a vulnerability is worth fixing [56, 57].

In-context security: Xie *et al.* [71] proposed a tool to remind web developers of secure programming practices in their Integrated Development Environment (IDE). The tool statically analyzes the code and alerts developers of potential issues on-the-spot. Although it does not cover all vulnerability types, usability evaluations (*e.g.*, [40, 59, 60, 72]) showed promising results in encouraging developers' attentiveness to security. Focusing on mobile applications, Nguyen *et al.* [44] developed an IDE plugin to support Android app developers adhere to and learn about security best practices. Studies suggest that the plugin significantly improves code security regardless of the developer's experience [44].

APIs and documentation: The use of Application Programming Interfaces (APIs) is recommended to improve code security [69]. However, further research is needed for improving the design of APIs to reduce vulnerability-causing mistakes [64] and account for security implications that may be missed by developers [46]. For example, Acar *et al.* [3] found usability issues in several cryptographic APIs that can result in compromised code security. In addition, many software security guidance resources available to developers lack helpful concrete examples, fail to address important topics, and some include obsolete advice [7]. This is an unfortunate finding, given that developers often rely on resources that are not necessarily ideal for security [4, 5, 24]. To partially address this, Gorski *et al.* [26] integrated context-sensitive security advice in a cryptographic API, which significantly improved the security of code using this API.

Overall, researchers have argued for more developer-centric security experiences, *e.g.*, by providing developers with practical security experience in using code analysis tools [13], facilitating security education in-context of developers' IDEs [47], focusing security training on addressing weaknesses in developers' security knowledge [61], and facilitating interaction between developers and security experts [61, 66].

Existing research focuses on helping developers improve their code security by reducing their cognitive load. However, several research gaps remain in addressing the human aspects of software security, such as factors that motivate developers to value and address security. In this paper, we take a human-centric approach to explore developers' software security strategies and motivation. In addition, we investigate different characteristics that may influence security processes, such as the development methodology, company size, and whether the development team employs TDD.

3 METHODOLOGY

We conducted an IRB-approved anonymous online survey with professional software developers using Qualtrics [1].

Survey Design. The survey included different types of questions, *e.g.*, multiple choice, Likert-scale, and short answer questions. The survey had two main sections, grouping questions by topic to minimize the cognitive load on participants and allow them to consider the topic more deeply [39]. The first section covered demographic information and investigated participants' general work motivation through the established 18-item Work Extrinsic and Intrinsic Motivation Scale (WEIMS) [62]. The second section focused on software security, specifically participants' efforts towards security, their strategies for handling security, and their opinions about their teams and experiences with security issues, as well as software security motivations and deterrents. In addition, we asked participants to describe what it means to them "to include security into the development process" to capture their original understanding of software security. However, to ensure that participants have a baseline understanding of software security, we then provided a brief explanation of software security and how it differs from security functions. Survey questions were informed by our previous qualitative research [8, 9, 30]. More details about the questions and format are available in Section 5, along with the corresponding results.

Testing the Survey Tool. We followed Dillman's recommended three-stage process [21] to pre-test the survey. First, the survey was reviewed by colleagues and experts in the field to uncover potential misunderstandings or unexpected outcomes. Next, we discussed the survey's clarity and motivation with developers. Finally, we performed pilot-testing with 11 developers to identify any flaws in the survey and to determine whether it is of appropriate length.

Participant Recruitment. Recruiting developers is one of the challenges of this type of research [6, 49]. To reach a wide range of developers, we recruited through two methods. (1) Through Qualtrics' [1] paid service; we paid Qualtrics $32 USD per participant for recruitment and data collection. Participants received the equivalent of $6.40 in gifts (*e.g.*, SkyMiles, gift cards). (2) Through announcing the survey to our professional and industry contacts; participants received a $10 Amazon gift card as compensation.

Data Quality. We took multiple precautions to ensure data quality. We provided participants with a description of software security to avoid confusion and differences in interpretation. Participants were prevented from progressing with the survey until they showed understanding of our description of software security. We discarded responses with less than seven minutes for survey completion time, and responses with invalid data, *e.g.*, gibberish in the open-ended question or conflicting responses.

Participant Demographics. Through the different channels, we recruited a total of 140 participants, and discarded 17 for quality issues. The data reported herein is from the remaining 123 valid responses. Average survey completion time was 24 minutes ($Md = 17$). Participant demographics are available in Table 1. Participants are currently working in development in Canada ($n = 63, 51\%$) or the US ($n = 60, 49\%$). They employ different development methodologies and develop a wide range of applications. The average company age where participants work is 41.3 years ($Md = 20$). Our dataset includes a good range of established companies: 25^{th} and 75^{th} percentile is 15 and 50 years, respectively.

4 SURVEY ANALYSIS

All the results presented in this paper represent participants' self-reported behaviours and attitudes. Data analysis for the open-ended question followed an inductive approach. The first author performed open coding, and both authors regularly discussed emerging themes and common patterns in the data. Quantitative data analysis used SPSS Statistics. All statistical tests assumed $p < .05$ as a significant level, unless Bonferroni-correction was applied.

All survey questions were optional, thus missing values may exist. These are ignored from the analysis, in which cases we indicate the actual number of data points (participants) when reporting the results.

Factor Analysis

We used factor analysis to analyze participants' security strategies, motivators, and deterrents. Principal axis factor analysis enabled us to group closely related information, thus, reducing the set of variables into a smaller set (*factors*), while retaining the majority of the original information [23].

Table 1: Summary of participant demographics

Country and Gender		
Canada		63 (51%)
USA		60 (49%)
Male		93 (76%)
Female		28 (23%)
Other or not specified		2 (2%)
Professional Experience		
Time spent in company		μ = 8 years (Md = 5)
Time spent in team		μ = 4.6 years (Md = 2.5)
Overall development experience		μ = 16.4 years (Md = 15)
Organization Information		
Company Age		μ = 41.3 years (Md = 20)
Size	1-249	34 (28%)
	250-999	29 (24%)
	1,000 or more	60 (49%)
Team Information		
Size		μ = 13.3 members (Md = 8)
TDD	Yes	32 (26%)
	No	82 (67%)
	Don't know	9 (7%)
*Dev Method	Waterfall development	27 (22%)
	Iterative (not truly agile)	26 (21%)
	Rational Unified Process	1 (1%)
	Agile development	58 (47%)
	Other	10 (8%)

*One participant did not indicate a development methodology.

Within and between subjects statistical tests on strategies, motivations, and deterrents all used the resultant factors.

As recommended, we retained variables with absolute factor loadings greater than 0.4 [23, 58]. For all factor analyses, we used the Kaiser Meyer-Olkin (KMO) measure [36, 37] to verify the sampling adequacy.

5 RESULTS

Developers' Work Motivation

To explore participants general work motivation, we generated the Work Self-Determination Index (W-SDI) [62] from the WEIMS. A positive W-SDI indicates a self-determined motivation profile, whereas a negative score indicates non-self determination [62]. Results indicate that our participants do not lack motivation with respect to performing their job; the vast majority (89%) exhibited self-determined motivation profiles (W-SDI > 0).

Developers' Mental Models of Software Security

65% of participants had a reasonable understanding of software security. Most participants discussed that software security aims to minimize vulnerabilities, minimize the negative consequences of malicious attacks, and prevent unauthorized access or use of their software or the data it handles. Participants also explained that security should be considered from the earliest stages and throughout the development process. For example, one participant described software security as, "*To think about security from the earliest planning phases as possible [...] and continue to focus on security implications throughout the remainder of the development process.*" In addition, some participants indicated that security defences should be proactive, and that developers should "*think secure from the beginning*" and adopt an attacker-mindset. For example, a participant said, "*rather than asking how will we achieve 'this', you ask how will someone exploit 'this'. [...] when your processes are done in a proper, security conscious way, as much of the potential harm as possible should be mitigated.*" Participants also discussed various methods to ensure software security, such as internal and external audits, security testing, automated checks, code analysis and reviews, thinking about security when writing code, and incorporating security in design. Some participants also discussed the importance of following best practices, using tools and programming languages approved by their organizations, and receiving support from security experts in their organizations.

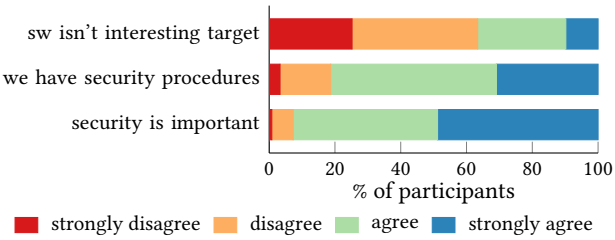

Figure 1: Participants' opinion of their teams.

Behaviours and Attitudes

Participants indicated on a 4-point Likert scale (1:strongly disagree to 4: strongly agree) their agreement with statements about their teams. As shown in Figure 1, participants generally indicated that their teams believe in the importance of software security and that they have specific procedures in place to address it, even though they mostly do not think that their applications are interesting targets for attackers. All participants, except one, who reported security is not important for their teams also indicated that their software is not an interesting target for attackers.

Experiencing Security Issues

On 5-point Likert scales, participants indicated their satisfaction with their teams' security processes and the likelihood that their software has vulnerabilities. Figure 2 shows that

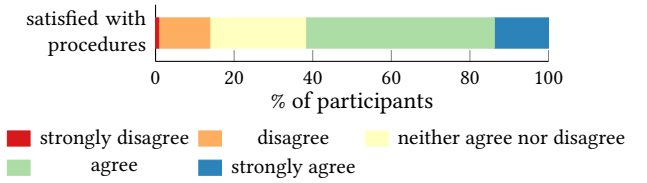

Figure 2: Satisfaction with teams' procedures

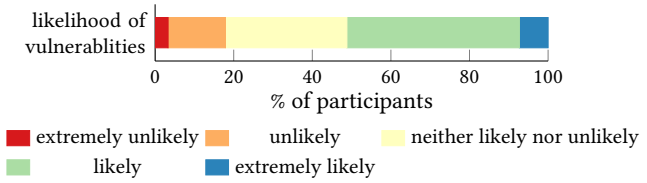

Figure 3: Likelihood of vulnerabilities in team's code

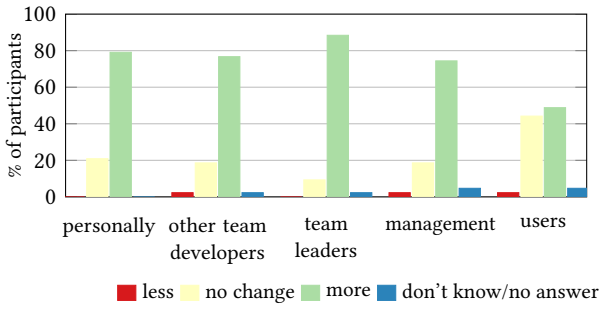

Figure 4: Longterm change in awareness and concern for security following experiences of security issues (n = 43).

in general, participants are satisfied with their team's handling of software security. However, despite their satisfaction, participants believed that software developed by their team likely contains security issues (Figure 3).

Participants were asked to report whether their software has ever experienced a security issue. More than a third of participants reported at least one security issue. Vulnerable shipped code was most frequently reported (24%) out of the three potential security issues in the survey. Fourteen percent of participants indicated that vulnerabilities were discovered before their software was shipped, and 11% reported their software experienced a security breach. We note that these numbers are not mutually exclusive; some participants (11%) reported multiple security issues.

For participants who reported security issues (n = 43), we explored the long-term reaction to experiencing such issues by the different stakeholders. Although it may be expected that awareness and attitude towards security improves right after experiencing an issue, our data suggests that this effect is longstanding. Figure 4 shows that 79% of participants indicated that experiencing a security issue increased their awareness and concern for security over the long-term. Participants also reported the same effect on other developers in their teams (77%), team leaders (88%), upper management (74%), and users (49%). This implies that experiencing a real threat can help avoid the optimism bias[1] [52, 65] and can lead to improved attitudes and behaviours towards security.

Forty-four percent of participants indicated that company security issue(s) did not change their users' awareness and concern for security. "Users" had the highest percentage of "no change" across the different stakeholders, as shown in Figure 4. This is reasonable given that users are not typically aware of such software security issues unless, *e.g.*, a security breach is publicized or users directly experience the effects.

We will now discuss our results arranged by research question. In reporting Likert-scale questions, we group "strongly agree" and "agree" responses within the text, and likewise group "strongly disagree" and "disagree" responses. We use S_i, M_i, and D_i as labels for statement relating to strategies, motivations, and deterrents in the survey.

RQ1: how software security fits in the SDLC.

The survey had several questions exploring development teams' efforts and strategies towards software security.

Efforts Towards Security. Participants reported the percentage of effort directed towards security out of the overall development lifecycle effort. They also reported the percentage of effort out of all security efforts as a percentage for different development stages (design, implementation, developer testing, code analysis, code review, and post-development testing). The total for all stages equaled 100%.

As shown in Figure 5, participants indicated that, on average, 19% (Md = 10%) of their teams' overall effort in the development lifecycle relates specifically to security tasks. Six participants (5%) indicated that their teams do not spend any effort on security.

We used Friedman's ANOVA to determine whether the distribution of security efforts significantly differs across the different SDLC stages. As Figure 5 shows, security effort in the implementation stage was significantly higher than in the code analysis, developer testing, code review, and post-development testing stages. Security effort in the design stage was also significantly higher than in the code analysis and code review stages. It is unclear why participants focused their efforts at these two stages; it could be because they try to get it right from the beginning, thus reducing the effort needed during later stages, or it could be because later stages are mainly functionality-oriented.

[1]Optimism bias is the belief that "misfortune will not strike me" [52, 65].

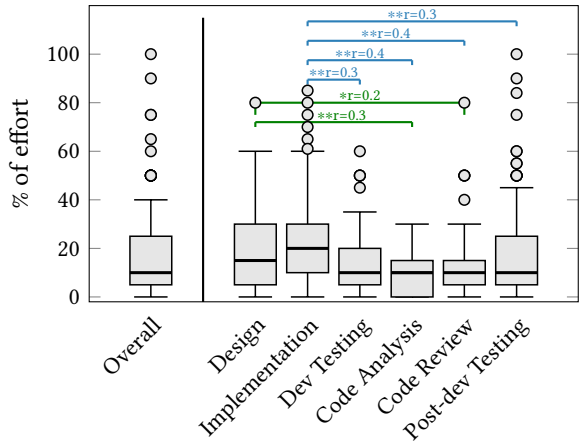

Figure 5: Software security efforts in the SDLC. (Figure shows stages that significantly differ in efforts towards security. $^{**}\chi^2_F(5) = 78.9, n = 123.$ $*: p < .05$, $**: p < .01$**)**

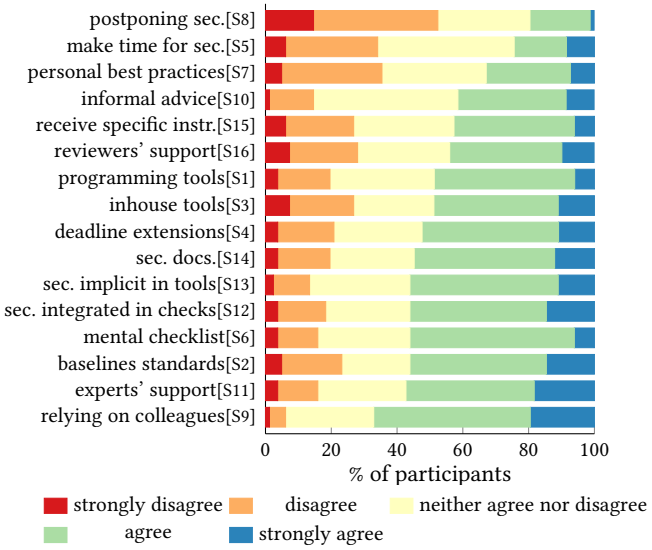

Figure 6: Strategies for handling software security ($n = 82$)

Strategies to Address Software Security. Participants rated their agreement with relying on 16 potential software security strategies on a 5-point Likert scale.

As shown in Figure 6, most participants indicated that when fixing a security issue, they rely on support from colleagues who faced similar issues. For security advice, the majority of participants reported relying on those with more experience. More than half also indicated relying on their own personally-devised security checklists to handle security, or on company-wide strategies, *e.g.*, automated checks.

We performed factor analysis to integrate these 16 strategies to a smaller set and found that 12 strategies could be grouped into two factors; four strategies did not conform to any factor (results in Table 2). We named the first resultant factor: *company-wide engagement*, as it describes how developers rely on their companies' strategies and support, *e.g.*, relying on the more experienced team members (conforming with previous research [14, 38]), or using custom tools that handle software security. This factor encompassed nine strategies. The second factor incorporated three strategies and is named: *personal strategies*, where developers devised their own software security strategies, *e.g.*, having their own mental checklist of issues to consider.

Table 2: Factor analysis for security strategies.

Variables (Strategies as presented in the survey)	factor loading
Company-wide Engagement ($\alpha = 0.9$)	
S13: Software security best practices are incorporated in tools we use	0.9
S12: Software security best practices are incorporated in automated checks we run	0.8
S2: Our company/team has baseline security standards with which 3rd party code should comply	0.8
S11: I can rely on the more experienced members of my company/team for help and security advice	0.8
S9: When working on a software security issue, I can get help from others who worked on similar issues	0.5
S3: We built our own in-house frameworks to help guarantee software security	0.5
S15: I receive specific instructions on how to solve security issues found in my code	0.5
S14: We have a document/checklist of items that we need to consider for our application to be secure	0.5
S16: In code reviews, reviewers explain security issues and fixes to me rather than referring me to resources/books	0.4
Personal Strategies ($\alpha = 0.6$)	
S6: I have my own mental checklist of software security issues that I need to consider in my code	0.9
S7: I have come up with my own software security best practices	0.7
S5: When a deadline approaches, I try to reduce my workload to focus on securing my software	0.5
Strategies not belonging to any factor	
S1: We rely on libraries and frameworks (including APIs) to help guarantee software security	
S4: I can get deadline extensions to handle software security	
S8: If I didn't have time to address software security, I'd ship the product after adding a work around that allows me to remotely disable the software feature suffering a security breach	
S10: I prefer to ask for software security advice informally (e.g., by casually asking a colleague, or through discussions over lunch)	

$KMO = 0.8$

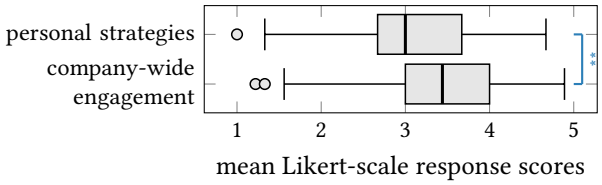

Figure 7: Use of strategies for handling software security after factor analysis ($n = 87$). (1:strongly disagree – 5:strongly agree. Significant difference is shown. $**: p < .01$)

For further analyses, we created a variable for each factor by averaging participants' response to all strategies belonging to the factor. Figure 7 shows that participants ($n = 87$) rely more on *company-wide engagement* than on their own *personal strategies* to handle software security. A Wilcoxon signed rank test confirmed this observation ($T = 814, p < .01, r = -0.3$). This affirms the importance of companies' role in ensuring a secure foundation for their software and promoting software security, *e.g.*, by building a security culture, and facilitating security learning opportunities.

RQ2: Security Motivators and Deterrents

To explore what motivates developers to address software security, we presented participants with a list of 21 potential motivators, as well as 29 statements that could explain reasons for deferring security. Participants ranked their agreement with each statement on a 5-point Likert scale.

Software Security Motivators. We asked participants *"I care about security because..."* and presented potential motivations for software security. In addition to the Likert scale, this question had a "not applicable" option in case a motivation did not apply to a participant's workplace.

As shown in Figure 8, the top six reasons to care about software security are self-driven motivations [53]. Participants are motivated by the challenge or by their own values (*e.g.*, to protect their users). Receiving financial rewards (an external motivation) was reportedly least motivating.

We used factor analysis to combine the 21 motivators into a smaller set (Table 3). Our factor analysis grouped 15 of them into four factors; six motivators did not conform to any particular factor. We named the factors: *workplace environment*, *identifying with security importance*, *rewards*, and *perceived negative consequences*. Out of the four factors, *rewards* is the only one representing external motivations [53]. For further analyses, we created a variable for each factor by averaging[2] participants' responses to all motivators belonging to the factor. Figure 9 shows participants' ($n = 76$)[3] motivations for software security.

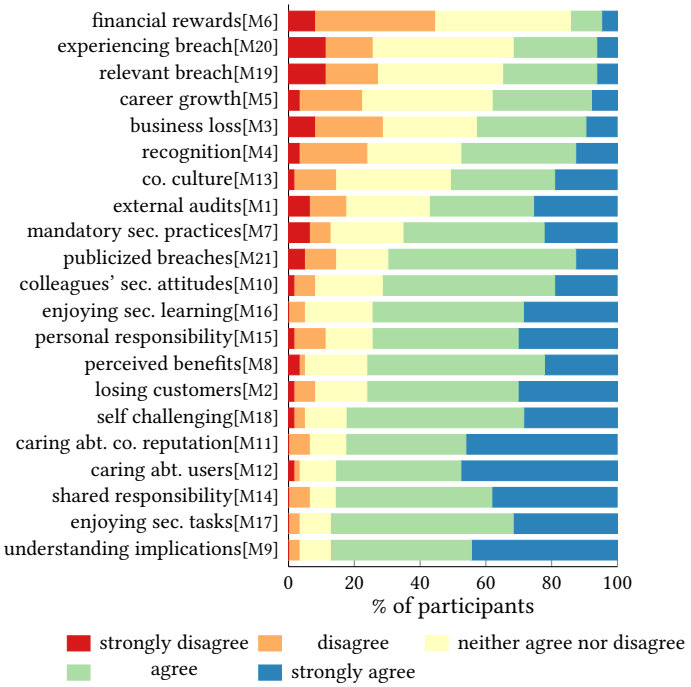

Figure 8: Software security motivators ($n = 63$)

We found statistically significant differences between the four software security motivators ($\chi^2_F(3) = 85.75, p < .01$). Pairwise comparisons using Wilcoxon tests with Bonferroni correction were used to follow up this finding. We found that *rewards* was the least significant motivator compared to *workplace environment* ($T = 1.13, p < .01, r = 0.44$), *identifying with security importance* ($T = 1.78, p < .01, r = 0.69$), and *perceived negative consequences* ($T = -0.95, p < .01, r = -0.37$). In addition, it appears that *identifying with security importance* is the most motivating factor; it can motivate developers more than *perceived negative consequences* ($T = 0.83, p < .01, r = 0.32$) and *workplace environment* ($T = -0.65, p < .05, r = -0.25$).

Participants' software security motivation appears to match their general work motivation pattern. Their top security motivators are all intrinsic and internal motivations [53].

Deterrents to Software Security. Participants generally opposed statements that imply deferring or ignoring security, as suggested by the overwhelmingly red and orange chart in Figure 10. The biggest deterrent to software security was the lack of a formal plan or process, followed by participants being unaware of security code-analysis tools.

[2]According to Boone and Boone [18], this approach is appropriate with ordinal data because we are interested in the "composite score" representing the factors.

[3]We could only include data from participants who answered all questions in each factor.

Table 3: Factor analysis for motivation

Variables (Motivators as presented in the survey)	factor loading
Workplace Environment (α = 0.9)	
M13: Software security is in my company's culture	0.7
M7: My company mandates security practices & I have to follow them	0.7
M10: My colleagues care about software security	0.6
M8: I see the benefit in security practices mandated by my company	0.6
Identifying with Security Importance (α = 0.8)	
M14: Software security is a shared responsibility by all those involved in the development lifecycle	0.8
M12: I care about my users' security and privacy	0.7
M9: I understand that my code can have sec implications	0.6
M16: I feel good when I learn about software security	0.6
M15: I see software security as my responsibility	0.6
M11: I care about my company's reputation	0.4
Rewards (α = 0.8)	
M6: My efforts towards sw sec are financially rewarding	0.8
M4: My efforts towards software security are recognized	0.8
M5: My efforts towards software security help me grow in the company	0.7
Perceived Negative Consequences (α = 0.6)	
M21: I realized securing my code is important after reading about security breaches in the news	0.7
M1: My company is audited for sw sec by an external entity	0.6
Motivations not belonging to any factor	
M2: My company would lose customers in case of a sw sec breach	
M3: My company could fail in case of a software security breach	
M17: I feel good when I address potential sec issues in my code	
M18: I like to challenge myself to write secure code	
M19: Similar software to that on which I work suffered a security breach and management now cares about securing our applications	
M20: Similar software to that on which I work suffered a security breach and it was an eye-opener for me	

$KMO = 0.9$

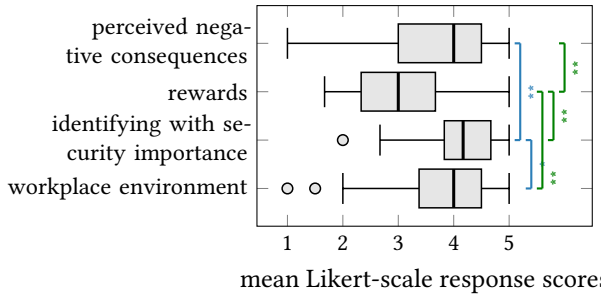

Figure 9: Motivations for software security after factor analysis (n = 76). (1:strongly disagree – 5:strongly agree. Significant difference is shown $*: p < .05$, $**: p < .01$)

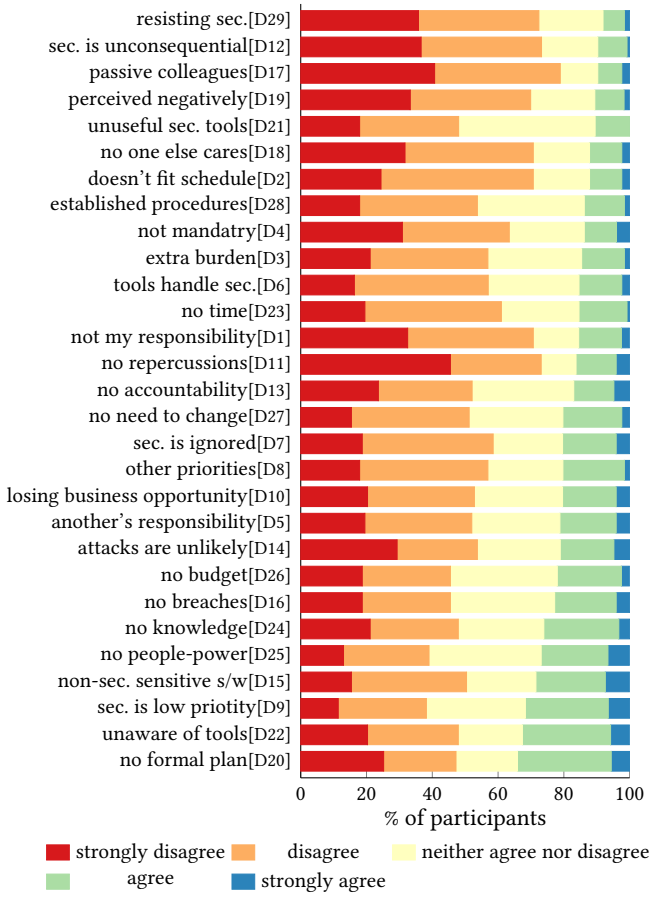

Figure 10: Deterrents to software security

Our factor analysis combined 18 of the 29 software deterrents into four factors; 11 deterrents did not correspond to any particular factor (Table 4). Our first two factors are *security is irrelevant* and *competing priorities & no plan*. These describe how a lack of security can stem from systemic causes within the company or team, such as whether there are consequences for the lack of security, whether security is a priority, and if specific security plans exist. The other two factors, *unequipped for security* and *disillusioned*, describe security deterrents on a more personal level, *e.g.*, a lack of support, knowledge, and awareness can deter developers from addressing security, as well as being in a workplace environment that thwarts, rather than nurtures, security efforts.

Considering the four factors, as Figure 11 shows, the two most frequent deterrents to software security were (1) being *unequipped for security* because of a perceived lack of security

Table 4: Factor analysis for security deterrents

Variables (Deterrents as presented in the survey)	factor loading
Security is Irrelevant ($\alpha = 0.8$)	
D1: Software security is not my responsibility because it's not in my job description	0.6
D15: The software I develop is not prone to security attacks	0.6
D16: Things are fine as they are, we haven't experienced any security breaches	0.6
D11: There are no repercussions to ignoring sw security	0.6
D12: We do not have competition, so we won't lose customers in case of a software security issue	0.5
D5: Sw sec is handled by someone else in the product lifecycle	0.5
Competing Priorities & no Plan ($\alpha = 0.9$)	
D20: We do not have a formal process for software security	-0.7
D4: Software security is not mandated by my employer	-0.7
D8: We defer software security due to competing priorities	-0.7
D7: My team doesn't spend any specific efforts towards software security	-0.6
D9: In my team, it is more important to deliver features on time than to address software security	-0.6
Unequipped for Security ($\alpha = 0.8$)	
D22: I am not aware of tools that would allow security analysis of my code	0.8
D24: I do not have necessary knowledge to address sw sec.	0.6
D21: Available security code analysis tools are not useful	0.5
D28: We have been following the same procedures for years and I don't want to change them	0.5
Disillusioned ($\alpha = 0.9$)	
D18: I understand the importance of addressing security, but I won't waste my time on it since no one else does	-0.7
D19: I used to push for software security, but I was perceived negatively by my colleagues	-0.7
D17: No one else cares about software security, I won't either	-0.6
Deterrents not belonging to any factor	
D2: Software security does not fit in my schedule	
D3: Sw sec is a burden on top of my main responsibilities	
D6: We don't have to worry much about security because frameworks [...] we use handle software security for us	
D10: If we focus more on software security, we might lose our business opportunities	
D13: I won't be blamed if a security issue is found in my code	
D14: It's unlikely that attackers will attack us	
D23: I do not have time to address software security	
D25: There aren't enough people in my team to address sw sec	
D26: My team does not have the budget to address sw sec	
D27: We're doing fine, I don't think we should change in terms of software security	
D29: I tend to resist when I get assigned a security task	

$KMO = 0.9$

knowledge or the unavailability of necessary tools, and (2) *competing priorities & no plan*, where security has a lower priority than other aspects of the software and the team lacks specific security plans or procedures.

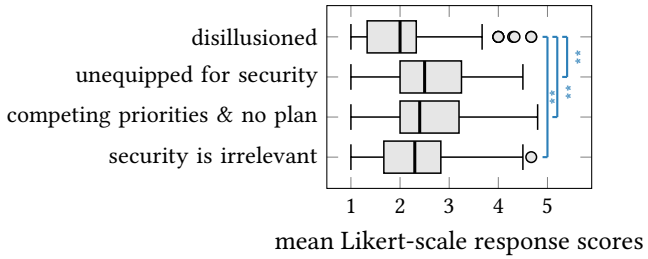

Figure 11: Security deterrents after factor analysis. (1:strongly disagree – 5:strongly agree. Significant difference between deterrents is shown. $* : p < .05$, $** : p < .01$)

We found a statistically significant difference between participants' responses for the four factors ($\chi^2_F(3) = 51.1, p < .01$). Pairwise comparisons using Wilcoxon tests with Bonferroni correction showed that being *disillusioned* was less likely than thinking *security is irrelevant* ($T = 0.7, p < .01, r = 0.3$), having *competing priorities & no plan* ($T = 0.9, p < .01, r = 0.3$), and being *unequipped for security* ($T = 1, p < .01, r = 0.4$). No other pairs showed significant differences.

6 RQ3: EFFECT OF DIFFERENT CHARACTERISTICS ON SOFTWARE SECURITY

In this section, we explore the impact of three main characteristics on software security overall: (1) the development methodology used by participants' teams, (2) the size of the company where participants work, and (3) whether they perform TDD. Specifically, we explore whether these characteristics influence security efforts, software security strategies, security motivators, or deterrents to software security. We focus on these three characteristics because they were considered as potential influencers on software security in previous literature (*e.g.*, [12, 15]) or in our previous discussions with software developers and security experts.

Development Methodology. Focusing on the three development methodologies with the highest percentages of participants in our data: Waterfall (22%), Iterative(21%), and Agile development(47%), our analysis using Kruskal-Wallis tests with Bonferroni-correction showed that, contrary to previous literature [12, 15, 54], the development methodology did not significantly influence teams' handling of software security. We did not find evidence that the development methodology influenced teams' overall effort towards software security, nor did it influence their effort per development stage. In addition, it had no influence on software security strategies or deterrents to security. Our results indicated that the development methodology may influence

some security motivations, *identifying with security importance* ($H(2) = 7, p < .05$), *rewards* ($H(2) = 6.4, p < .05$), and *perceived negative consequences* ($H(2) = 6.4, p < .05$). However, follow-up pairwise comparisons with Bonferroni-correction were not significant.

Company Size. Following the classification used in North America [16, 27], we classified participants' companies into either Small and Medium Enterprises (SMEs) if the company had fewer than 500 employees, and Large Enterprises (LEs) otherwise. Our dataset contained 49 (40%) participants in SMEs and 74 (60%) in LEs.

Using Mann-Whitney tests, we found no evidence that the company size influenced the percentage of effort on software security, overall or per development stage. In addition, it did not influence participants' software security strategies.

However, our results show a significant difference in security motivations between SME and LE participants. Being in a *workplace environment* that nurtures security was more motivating for participants in LEs, compared to those in SMEs ($U = 861, n = 76, p < 0.05, r = -0.2$).

Our results also show that deterrents to software security vary significantly with company size. Specifically, having *competing priorities & no plan* is significantly more common deterrent to security for participants in SMEs compared to those in LEs ($U = 1345.5, p < 0.05, r = -0.2$). Likewise, being *unequipped for security* is a significantly more common deterrent to security for SME participants compared to their counterpart ($U = 1413, p < 0.05, r = -0.1$).

We also performed post-hoc analysis to explore further effects of company size, *e.g.*, whether it had an effect on participants' behaviours and attitudes towards software security, or whether the company experienced security issues. All the tests were not significant, *e.g.*, we found no difference between SME and LE participants in considering that their applications are interesting targets for attackers ($U = 1552.2, n = 123, p = 0.2$).

Test-Driven Development (TDD). Out of the characteristics explored, TDD most influences software security.

Our results show that efforts directed towards software security are influenced by whether the team performs TDD. Participants who perform TDD spend significantly more overall effort on security than those who do not perform TDD ($U = 905, n = 114, p < 0.01, r = -0.2$). Focusing on each SDLC stage, TDD participants spend significantly higher efforts towards security during code analysis than their counterparts ($U = 554.5, n = 114, p < 0.01, r = -0.3$).

We also found that adopting TDD influences software security strategies. TDD participants rely significantly more on *company-wide engagement* ($U = 397, n = 80, p < 0.01, r = -0.3$) and on their own *personal strategies* ($U = 506, n = 80, p < 0.05, r = -0.2$) to handle software security than participants who do not perform TDD.

Finally, our results show that TDD participants are not significantly different than those who do not perform TDD when it comes to security deterrents and the majority of security motivators. However, we found that *rewards* is a more significant security motivator to TDD participants compared to their counterparts ($U = 453, n = 68, p < 0.01, r = -0.1$).

7 DISCUSSION

Many participants indicated their companies faced security issues, including security breaches. This could be because functionality and on-time shipping were prioritized and security was postponed. In fact, seven participants who reported vulnerabilities in shipped code indicated that when deadlines approach, they ship their code with a backdoor to address the security issues later. This behaviour is clearly troubling.

However, in general, our results are promising for software security. Whereas previous research [67, 70, 73] found that developers generally exhibit a "security is not my responsibility" attitude, the vast majority of our participants acknowledge the importance of software security and have specific procedures in place to address it. The few participants who indicated security is not important for their teams indicated that their software is not an interesting target to attackers. We do not imply that completely ignoring security is acceptable, but rather consider the possibility that these teams may be making an *educated* economic decision, having assessed the risk and found that it was negligible.

In addition, our participants appear to have a general understanding of what software security means, and they acknowledged that their applications may have security issues despite their efforts. It is interesting that some participants ($n = 33$) indicated that security is important for their teams and that they are satisfied with how they are handling software security, but also indicated that their software is likely vulnerable. We did not expect this combination. One explanation could be that participants were being pragmatic; there will always be security issues and you can never prove security [32]. Another explanation could be that participants are satisfied that they are doing their best to ensure security given their circumstances, even if it may not be ideal or enough. Previous research [9] discussed that a lack of resources may discourage teams from addressing security or following security best practices. In our work, we found evidence that the lack of resources may be why participants believe that their applications are vulnerable, despite being satisfied with their practices. For example, 45% (15 of 33) of participants displaying this interesting combination indicated they lack at least one of: knowledge, awareness, budget, tools, time, and people-power to handle software security. In addition, the lack of security plans and resources were also reasons for deferring security at SMEs. These are reasonable reasons that may prevent teams or developers from focusing

on software security. By identifying these deterrents, we can better focus our efforts on overcoming them, *e.g.*, through providing better support for teams to devise security plans that fit their resources and work styles.

In general, our participants were self-motivated towards their work, as well as software security. Being a hard and cognitively demanding task [28, 47], software security would likely benefit from developers' self-motivation. Research showed that this type of motivation leads to better performance, engagement, and cognitive abilities [53]. Thus, rather than relying mainly on external motivations (*e.g.*, rewards), companies could focus their efforts on promoting internal motivations towards software security (*e.g.*, by portraying security as their collective professional responsibility, and raising developers' awareness of the implications of their code on their users and their company's reputation). However, we should not expect developers to take on the challenge without adequate support. In addition to security tools and methodologies, developers should receive support within their workplace. For example, companies can work towards establishing security plans to guide developers' security efforts. Companies and teams can also facilitate collaboration between developers and security experts. This collaboration can bridge the gap between the two groups [61], and would enable developers gain practical security experience which can improve their code.

8 LIMITATIONS

Conducting the survey online may have influenced data quality, but we took measures to filter out poor quality responses. The different methods of recruitment and compensation helped reach a broad range of participants, though, this difference may have influenced developers' willingness to answer the survey. Our results are based on participants' self-reported responses, which may be subject to bias and may not exactly represent real-life. However, we followed recommendations to reduce social-desirability bias by ensuring participants' anonymity [43]. The lists of software security strategies, motivations, and deterrents included in our survey are non-comprehensive. However, our lists reflect at least a subset of existing developers' strategies, motivations and deterrents to software security [30].

9 CONCLUSION AND FUTURE WORK

We presented a survey study with 123 participants to explore how they address software security, as well as security motivators and deterrents. Participants consider security as part of their development process to varying degrees. Most interestingly, we believe that our results affirm that *developers are not the weakest link*. Our analysis shows that participants are self-driven in their work in general, as well as in their motivation towards software security. Thus, developers in our study are not explicitly ignoring security, dismissing it, or considering it outside of their responsibility. In fact, they are most motivated towards software security when they recognize and identify with its importance. On the other hand, the most important deterrents for software security relate to the (mis)management of the process. For example, dealing with competing priorities, and the lack of security plans, procedures, knowledge, or resources are the main causes for deferring security. Our work highlights the need to look beyond the individual and to focus on understanding organizational issues that lead to insecure practices.

For future work it would be interesting to explore potential relationships between motivations, deterrents, and strategies for software security (*e.g.*, do certain deterrents lead developers to adopt certain strategies?) In addition, as our results indicated that experiencing a security issue (*e.g.*, a breach) can increase software security awareness, it would be interesting to investigate security procedures and attitudes in companies that have experienced such issues and compare it to others that have not.

10 ACKNOWLEDGMENTS

We thank our participants for their time. H. Assal acknowledges her NSERC Postgraduate Scholarship (PGS-D). S. Chiasson acknowledges funding from NSERC for her Canada Research Chair and Discovery grants.

REFERENCES

[1] [n. d.]. Qualtrics. https://www.qualtrics.com. [Accessed June-2018].
[2] [n. d.]. Risk Management Guide for Information Technology Systems. *NIST Technical Series Publication* ([n. d.]).
[3] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. 2017. Comparing the Usability of Cryptographic APIs. In *IEEE Symposium on Security and Privacy*.
[4] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *IEEE Symp. on Security and Privacy*. https://doi.org/10.1109/SP.2016.25
[5] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. 2017. How Internet Resources Might Be Helping You Develop Faster but Less Securely. *IEEE Security Privacy* 15, 2 (2017).
[6] Y. Acar, S. Fahl, and M. L. Mazurek. 2016. You are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users. In *IEEE Cybersecurity Development*. https://doi.org/10.1109/SecDev.2016.013
[7] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl. 2017. Developers Need Support, Too: A Survey of Security Advice for Software Developers. In *Cybersecurity Development (SecDev)*.
[8] H. Assal and S. Chiasson. 2018. Motivations and Amotivations for Software Security. In *SOUPS Workshop on Security Information Workers (WSIW)*. USENIX Association.
[9] H. Assal and S. Chiasson. 2018. Security in the Software Development Lifecycle. In *Symp. on Usable Privacy and Security*. USENIX.
[10] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh. 2008. Using Static Analysis to Find Bugs. *IEEE Software* 25, 5 (2008). https://doi.org/10.1109/MS.2008.130

[11] B. K. Marshall. [n. d.]. Passwords Found in the Wild for January 2013. http://blog.passwordresearch.com/2013/02/. [Accessed April-2017].

[12] D. Baca, M. Boldt, B. Carlsson, and A. Jacobsson. 2015. A Novel Security-Enhanced Agile Software Development Process Applied in an Industrial Setting. In *Int. Conf. on Availability, Reliability and Security*.

[13] D. Baca, K. Petersen, B. Carlsson, and L. Lundberg. 2009. Static Code Analysis to Detect Software Security Vulnerabilities - Does Experience Matter?. In *Int. Conf. on Availability, Reliability and Security*.

[14] R. Balebako and L. Cranor. 2014. Improving App Privacy: Nudging App Developers to Protect User Privacy. *IEEE Security Privacy* 12, 4 (2014).

[15] S. Bartsch. 2011. Practitioners' Perspectives on Security in Agile Development. In *Int. Conf. on Availability, Reliability and Security*. https://doi.org/10.1109/ARES.2011.82

[16] G. Berisha and J. Shiroka Pula. 2015. Defining Small and Medium Enterprises: A Critical Review. *Academic Journal of Business, Administration, Law and Social Sciences* 1 (2015).

[17] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler. 2010. A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World. *Commununications of the ACM* 53, 2 (2010).

[18] Harry N Boone and Deborah A Boone. 2012. Analyzing likert data. *Journal of extension* 50, 2 (2012), 1–5.

[19] CERT and CMU. [n. d.]. Cybersecurity Engineering. https://www.cert.org/cybersecurity-engineering/. [Accessed Feb-2017].

[20] B. Chess and G. McGraw. 2004. Static Analysis for Security. *IEEE Security & Privacy* 2, 6 (2004). https://doi.org/10.1109/MSP.2004.111

[21] D. A. Dillman. 2000. *Mail and Internet Surveys: The tailored design method*. John Wiley & Sons, Inc.

[22] EQUIFAX. 2018. 2017 Cybersecurity Incident & Important Consumer Information. https://www.equifaxsecurity2017.com. [Accessed June-2018].

[23] A. Field. 2013. *Discovering statistics using IBM SPSS statistics*. SAGE Publications Ltd.

[24] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl. 2017. Stack Overflow Considered Harmful? The Impact of Copy Paste on Android Application Security. In *IEEE Symp. on Security and Privacy*. https://doi.org/10.1109/SP.2017.31

[25] S. Garfinkel and H. R. Lipford. 2014. Usable Security: History, Themes, and Challenges. *Synthesis Lectures on Information Security, Privacy, and Trust* 5, 2 (2014).

[26] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. 2018. Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, 265–281. https://www.usenix.org/conference/soups2018/presentation/gorski

[27] Government of Canada. 2018. SME Research and Statistics. http://www.ic.gc.ca/eic/site/061.nsf/eng/Home. [Accessed June-2018].

[28] M. Green and M. Smith. 2016. Developers are Not the Enemy!: The Need for Usable Security APIs. *IEEE Security Privacy* 14, 5 (2016). https://doi.org/10.1109/MSP.2016.111

[29] G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, and L. Mounier. 2016. Toward Large-Scale Vulnerability Discovery Using Machine Learning. In *ACM Conf. on Data and Application Security and Privacy*. 12. https://doi.org/10.1145/2857705.2857720

[30] H. Assal. 2018. *The Human Dimension of Software Security and Factors Affecting Security Processes*. Carleton University.

[31] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel. 2008. Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses. In *IEEE Symp. on Security and Privacy (SP)*. https://doi.org/10.1109/SP.2008.31

[32] C. Herley and P. C. v. Oorschot. 2017. SoK: Science, Security and the Elusive Goal of Security as a Scientific Pursuit. In *IEEE S & P*. https://doi.org/10.1109/SP.2017.38

[33] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo. 2004. Securing Web Application Code by Static Analysis and Runtime Protection. In *WWW*. ACM, 13. https://doi.org/10.1145/988672.988679

[34] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *35th International Conference on Software Engineering (ICSE)*. 672–681. https://doi.org/10.1109/ICSE.2013.6606613

[35] N. Jovanovic, C. Kruegel, and E. Kirda. 2006. Pixy: a static analysis tool for detecting Web application vulnerabilities. In *IEEE S & P*. https://doi.org/10.1109/SP.2006.29

[36] H. F. Kaiser. 1970. A Second Generation Little Jiffy. *Psychometrika* (1970). https://doi.org/10.1007/BF02291817

[37] H. F. Kaiser and J. Rice. 1974. Little Jiffy, Mark IV. *Educational and Psychological Measurement* 34, 1 (1974). https://doi.org/10.1177/001316447403400115

[38] T. D. LaToza and B. A. Myers. 2010. On the Importance of Understanding the Strategies That Developers Use. In *CHASE*. ACM, 4. https://doi.org/10.1145/1833310.1833322

[39] J. Lazar, J. H. Feng, and H. Hochheiser. 2010. *Research methods in human-computer interaction*. John Wiley, Hoboken, NJ.

[40] H. Lipford, T. Thomas, B. Chu, and E. Murphy-Hill. 2014. Interactive Code Annotation for Security Vulnerability Detection. In *ACM SIW*. 6. https://doi.org/10.1145/2663887.2663901

[41] Microsoft Corp. [n. d.]. Microsoft Security Development Lifecycle. https://www.microsoft.com/en-us/sdl. [Accessed June-2016].

[42] A. Naiakshina, A. Danilova, C. Tiefenau, and M. Smith. 2018. Deception Task Design in Developer Password Studies: Exploring a Student Sample. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS)*. USENIX Association, Baltimore, MD, 297–313. https://www.usenix.org/conference/soups2018/presentation/naiakshina

[43] Anton J Nederhof. 1985. Methods of coping with social desirability bias: A review. *European journal of social psychology* 15, 3 (1985), 263–280.

[44] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl. [n. d.]. A Stitch in Time: Supporting Android Developers in WritingSecure Code. In *Conf. on Computer and Communications Security*. ACM, 13. https://doi.org/10.1145/3133956.3133977

[45] V. Okun, A. Delaitre, and P. E. Black. 2013. Report on the Static Analysis Tool Exposition (SATE) IV. In *NIST Special Publication 500-297*.

[46] D. Oliveira, T. Lin, M. Rahman, R. Akefirad, D. Ellis, E. Perez, R. Bobhate, L. DeLong, J. Cappos, and Y. 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

[47] D. Oliveira, M. Rosenthal, N. Morin, K.-C. Yeh, J. Cappos, and Y. Zhuang. 2014. It's the Psychology Stupid: How Heuristics Explain Software Vulnerabilities and How Priming Can Illuminate Developer's Blind Spots. In *ACSAC*. ACM, 10. https://doi.org/10.1145/2664243.2664254

[48] OWASP. [n. d.]. OWASP Guide Project. https://www.owasp.org/index.php/Category:OWASP$_{\mathrm{G}}$uide$_{\mathrm{P}}$roject. [Accessed Feb-2017].

[49] O. Pieczul, S. Foley, and M. E. Zurko. 2017. Developer-centered Security and the Symmetry of Ignorance. In *NSPW*. ACM, 11. https://doi.org/10.1145/3171533.3171539

[50] J. Radcliffe. 2011. Hacking Medical Devices for Fun and Insulin: Breaking the Human SCADA System. https://media.blackhat.com/bh-us-11/Radcliffe/BH$_{\mathrm{U}}$S$_{1}$1$_{\mathrm{R}}$adcliffe$_{\mathrm{H}}$acking$_{\mathrm{M}}$edical$_{\mathrm{D}}$evices$_{\mathrm{W}}$P.pdf. [Accessed

Feb-2017].

[51] Rapid 7 Community. 2015. #IoTsec Disclosure: 10 New Vulnerabilities for Several Video Baby Monitors. https://community.rapid7.com/community/infosec/blog/2015/09/02/iotsec-disclosure-10-new-vulns-for-several-video-baby-monitors. [Accessed Feb-2017].

[52] H.-S. Rhee, Y. U. Ryu, and C.-T. Kim. 2012. Unrealistic optimism on information security management. *Computers & Security* (2012). https://doi.org/10.1016/j.cose.2011.12.001

[53] R. M. Ryan and E. L. Deci. 2000. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist* 55, 1 (2000).

[54] R. Sass. 2016. How to Balance Between Security and Agile Development the Right Way. https://resources.whitesourcesoftware.com/blog-whitesource/how-to-balance-between-security-and-agile-development-the-right-way. [Accessed May-2018].

[55] R. Seacord. 2011. Top 10 secure coding practices. https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices. [Accessed Feb-2017].

[56] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford. 2015. Questions Developers Ask While Diagnosing Potential Security Vulnerabilities with Static Analysis. In *JESEC/FSE*. ACM, 12. https://doi.org/10.1145/2786805.2786812

[57] J. Smith, B. Johnson, E. Murphy-Hill, B. T. Chu, and H. Richter. 2018. How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool. *IEEE Transactions on Software Engineering* (2018). https://doi.org/10.1109/TSE.2018.2810116

[58] J. P. Stevens. 2002. *Applied multivariate statistics for the social sciences*. New Jersey: Lawrance Erlbaum Association.

[59] T. Thomas, B. Chu, H. Lipford, J. Smith, and E. Murphy-Hill. 2015. A study of interactive code annotation for access control vulnerabilities. In *IEEE Symp. on Visual Languages and Human-Centric Computing*. https://doi.org/10.1109/VLHCC.2015.7357200

[60] T. W. Thomas, H. Lipford, B. Chu, J. Smith, and E. Murphy-Hill. 2016. What Questions Remain? An Examination of How Developers Understand an Interactive Static Analysis Tool. In *Symp. on Usable Privacy and Security (SOUPS)*. USENIX Association.

[61] T. W. Thomas, M. Tabassum, B. Chu, and H. Lipford. 2018. Security During Application Development: An Application Security Expert Perspective. In *Conf. on Human Factors in Computing Systems*. ACM, Article 262, 12 pages. https://doi.org/10.1145/3173574.3173836

[62] M. A. Tremblay, C. M. Blanchard, S. Taylor, L. G. Pelletier, and M. Villeneuve. 2009. Work Extrinsic and Intrinsic Motivation Scale: Its value for organizational psychology research. *Canadian Journal of Behavioural Science* 41, 4 (2009).

[63] O. Tripp, S. Guarnieri, M. Pistoia, and A. Aravkin. 2014. ALETHEIA: Improving the Usability of Static Security Analysis. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 13. https://doi.org/10.1145/2660267.2660339

[64] S. Türpe. 2016. Idea: Usable Platforms for Secure Programming–Mining Unix for Insight and Guidelines. In *Engineering Secure Software and Systems*. Springer Int. Publishing.

[65] N. D. Weinstein and W. M. Klein. 1996. Unrealistic Optimism: Present and Future. *Journal of Social and Clinical Psychology* (1996).

[66] C. Weir, A. Rashid, and J. Noble. 2017. I'd Like to Have an Argument, Please: Using Dialectic for Effective App Security. *European Workshop on Usable Security (EuroUSEC)* (2017).

[67] J. Witschey, S. Xiao, and E. Murphy-Hill. 2014. Technical and Personal Factors Influencing Developers' Adoption of Security Tools. In *ACM Workshop on Security Information Workers (SIW)*. 4. https://doi.org/10.1145/2663887.2663898

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

[69] G. Wurster and P. C. van Oorschot. 2008. The Developer is the Enemy. In *New Security Paradigms Workshop (NSPW)*. ACM, 9.

[70] S. Xiao, J. Witschey, and E. Murphy-Hill. 2014. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. In *CSCW*. ACM, 12. https://doi.org/10.1145/2531602.2531722

[71] J. Xie, B. Chu, H. R. Lipford, and J. T. Melton. 2011. ASIDE: IDE Support for Web Application Security. In *Annual Computer Security Applications Conference (ACSAC)*. ACM, 10. https://doi.org/10.1145/2076732.2076770

[72] J. Xie, H. Lipford, and B.-T. Chu. 2012. Evaluating Interactive Support for Secure Programming. In *CHI Conference on Human Factors in Computing Systems*. ACM, 10. https://doi.org/10.1145/2207676.2208665

[73] J. Xie, H. R. Lipford, and B. Chu. 2011. Why do programmers make security errors?. In *VL/HCC*. IEEE.

[74] F. Yamaguchi, F. Lindner, and K. Rieck. 2011. Vulnerability Extrapolation: Assisted Discovery of Vulnerabilities Using Machine Learning. In *USENIX Conference on Offensive Technologies (WOOT)*. 1.