Influences of developers' perspectives on their engagement with security in code

Irum Rauf, Tamara Lopez, Helen Sharp, Marian Petre, Thein Tun
The Open University, UK
firstname.lastname@open.ac.uk

Mark Levine
John Towse
University of Lancaster
University of Lancaster, UK

Dirk van der Linden
Department of Computer and Information Sciences, Northumbria University, UK

Awais Rashid
Bristol Security Group, University of Bristol, UK

Bashar Nuseibeh
The Open University, UK
Lero, Republic of Ireland

Abstract

Background: Recent studies show that secure coding is about not only technical requirements but also developers' behaviour.

Objective: To understand the influence of socio-technical contexts on how developers attend to and engage with security in code, software engineering researchers collaborated with social psychologists on a psychologically-informed study.

Method: In a preregistered, between-group, controlled experiment, 124 developers from multiple freelance communities, were primed toward one of three identities, following which they completed code review tasks with open-ended responses. Qualitative analysis of the rich data focused on the attitudes and reasoning that shaped their identification of security issues within code.

Results: Overall, attention to code security was intermittent and heterogeneous in focus. Although social identity priming did not significantly change the code review, qualitative analysis revealed that developers varied in how they noticed issues in code, how they addressed them, and how they justified their choices.

Conclusion: We found that many developers do think about security – but differently from one another. Hence, effective interventions to promote secure coding must be appropriate to the individual development context. Data is uploaded at: https://osf.io/3jvrk/files/

CCS Concepts

• **Security and privacy → Social aspects of security and privacy**.

ACM Reference Format:
Irum Rauf, Tamara Lopez, Helen Sharp, Marian Petre, Thein Tun, Mark Levine, John Towse, Dirk van der Linden, Awais Rashid, and Bashar Nuseibeh. . Influences of developers' perspectives on their engagement with security in code. In *Proceedings of CHASE '22: Proceedings of the 15th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE) (CHASE 2022).* ACM, New York, NY, USA, 10 pages.

CHASE 2022, May 21–22, 2022, Pittsburgh, PA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

1 Introduction

The security of software is often considered a technical feat that developers need to accomplish by writing secure code. Recent studies show that writing secure code – that is, following coding practices that ensure that the software does not contain known vulnerabilities – is not only about the technical requirements of secure coding but also concerns developers' psychological barriers [28]. Thus, the development of secure code is not only about the quality of code *per se*, but also about the frame of reference adopted by those who are responsible for its implementation.

Additionally, developers are increasingly escaping organizational boundaries and working under their own priorities and constraints, with varying skills and experiences [8]. This diversity of developers is at the heart of a range of innovations in the digital economy. The software these developers produce can be, and is, deployed across systems pervasive in many aspects of human activity and is used by a global user base. Little is known about the security implications of freelance developers' software and security perspectives.

As part of an ongoing research collaboration between software engineering researchers and behavioral scientists, we are investigating what might influence developers' secure coding practices. Recognising the value and importance of more transparent open scholarship for software engineering [21], we developed and pre-registered [1] a psychological experiment in which we manipulated the social identity focus of 124 developers.

The study was structured as a between-group controlled experiment and included rich data collection to support analysis of the attitudes and reasoning that shaped the observed behavior. The design of the study was motivated by the Social Identity (SI) theory [15], the product of four decades of work in social psychology on the fluidity and complexity of group identity and its impact on real-world behaviours. The study aimed to assess the behaviour of the participants under conditions that prime them to think of themselves as individuals, or as part of the community mediated by the feelings of social concerns and responsibilities. The study

[1]The final design of the study was pre-registered at the OSF (Open Science Framework) https://osf.io/3jvrk/ to avoid any biases of researchers during the study and to strengthen the hypothesis-testing.

includes code-review tasks with open-ended responses to assess participants' engagement with code, and responses to psychological Likert-scale questions to assess developers' attitudes.

The study investigated two key research questions:

RQ 1: *Does the social priming of the participants, i.e., cueing them to think of themselves as part of the developer community, influence their security engagement with code?* This question was planned in the pre-registration of the study. To answer this question, we collected evidence of developers' security engagement from their open-ended responses for their code review tasks. We scored the evidence and conducted a quantitative evaluation of the control and primed groups – discussed in Section 3.2.

RQ 2: *How do developers attend to security in code?* The qualitative analysis of participants' responses was also part of pre-registration. To address RQ2, we conducted an inductive analysis [32] of participants' responses that offered evidence of engagement with security during code review. We investigated how participants identified security issues in code, how participants mitigated secure solutions in code, and why developers made decisions that involved security when they engaged with code – discussed in Section 4.

Our findings suggest that developers often think about security, but differ from each other in how they perceive security and address it (sometimes unknowingly). By providing developers an open canvas to register how they see code, we elicited a richer picture of how code is viewed by different developers.

Further, the study highlighted that the variations in developers' security orientation in the presence of a multitude of factors make studying a single factor (such as social identity priming) a difficult task. Nevertheless, we argue that more empirical studies need to consider developers' socio-technical context and take note of how developers identify with their actions, which can have implications for maintenance of their actions [33], [35].

The rest of the paper is structured as follows: Section 2 presents the study design. Section 3 and 4 present the quantitative and qualitative analysis, respectively. Section 5 discusses the insights gained from the study, and Section 6 concludes the paper.

2 The Experiment

The study was structured as a three-condition, between-subjects experiment. It was approved by the first author's university ethics committee. The design of the experiment builds on the SI theory [15] and focuses primarily on the influence of participants' identities on their (secure) coding behavior. It uses three-things manipulation technique [15] to prime the participants, i.e., the salient identity was primed by asking participants a number of questions about things they do: thinking about themselves personally (for personal identity) and thinking of themselves as part of the developer community (for social identity). The prediction was that a social identity will lead (on average) to decision making that is more security oriented. This hypothesis was predicated on work in social psychology (e.g., [18]) showing that, when social identity is salient, people have more concern for the welfare of others in the group – and hence they will be more concerned to avoid 'damaging' other developers through poor security practice.

The null hypotheses of the study were:

H0: There is no difference in security engagement with code between participants inducted into an identity framework (either personal or social) and those inducted into a baseline.

H_a : Participants inducted into an identity framework will be more aware of code security concerns than participants inducted into a baseline / non-identity framework.

H_1 : Participants inducted into a social identity framework will behave differently from participants inducted into a personal identity framework with respect to coding concerns (relevant to security).

2.1 Study Design

The study was presented as a 20-to 30-minute online experiment using the Qualtrics research tool. Three researchers, familiar with programming, attempted the online study to verify its timings.

Two code review tasks were prepared as behavioral tasks using code snippets from securecodewarrior.com [2] that had known vulnerabilities. Code reviews are "a manual inspection of source code by developers other than the author" (p.1, [17]). These tasks were presented to developers of each experimental group, and their order was counterbalanced for each group. The purpose of the behavioral tasks was to capture relevant developer behavior. To achieve good ecological validity, we were interested in capturing developers' behavior 'in the moment', along with evidence of security in their thinking when they look at code. It also replicated the social psychology behavior study reported by Levine et al.[18] to investigate the identity behavior among individuals in action. Recent work by Braz et al. [7] and Danilova et al. [11] also considers code reviews a good candidate to study developers' security behavior. Hence, we opted for code-review tasks that prompted developers with different programming skill levels to think of their coding behavior in their own way and reflect on it. The details are available online [3].

The code review tasks were written in Python with the Django web framework and were not modified from their original source. Some configuration files and other unnecessary files were removed due to time-limitations. Participants were told to assume that the code has no compilation errors and has all the required permissions to execute. Participants were asked to review the code and answer four open-ended questions: what does the code do, do they notice anything in particular about the code, how they can improve the code, and why they would make the change they suggest.

At the start of the study, after the information sheet and consent form, participants were primed for social identity, personal identity, or a control condition using the three things manipulation technique [15]. The code review tasks were then presented, followed by 8 Likert-scale (range 1 to 7) manipulation check questions: a four-item social identification check and four potential mediator questions ([27], [12][13]). We expected that, in the case of successful priming of participants in the social identity condition, participants would score higher in the social identification check and potential-mediator questions regarding any possible influence of responsibility and reputation.

Toward the end of the online study, participants were asked whether they were aware of the OWASP list of security vulnerabilities and to discuss them briefly. This was asked to ascertain how well-acquainted developers were with common security knowledge.

[2] Secure code warrior is "an integrated platform that provides secure coding training and tools" to developers.http://www.securecodewarrior.com

[3] https://bit.ly/3m9qyAl

At the end, demographic questions were asked, and participants were debriefed about code snippets and their source.

2.2 Pilot Study

A pilot study was carried out with 9 participants (3 in each group) with medium-to-high programming skill (as reported by the participants) recruited through personal contacts. Participants were advised to think aloud and ask any questions. The pilot study identified issues that were fixed in the final presentation of the study. In particular, participants not only noticed the vulnerabilities seeded by the secure code training platform, but also other security issues present in the code. Hence, we updated our code book and scoring to include other security issues.

2.3 Study Participants

The target sample was approximately 120 participants (approximately 40 participants per identity manipulation), in order to get a reasonable effect size. G*Power[4] indicated group N=32 was sufficient for acceptable power ≥ 0.8 with χ^2 for large effect sizes ≥ 0.5. Accounting for potential difficulties in recruiting participants and the potential need to filter and pre-process low quality answers, we estimated that an N=40 would allow for sufficient statistical power. We aimed to recruit participants within a 3-month time frame.

All the study participants were freelancers. To triangulate the data sources [31], we recruited participants from more than one channel. We collected data through two online crowdsourcing communities: upwork.com and freelancer.com. In total, we collected 124 valid responses: 82 from upwork.com and 42 from freelancer.com. Previous studies have reported recruiting freelancers through online communities as a viable option. They offer flexibility and access to wider population [38] and are also considered dependable [22].

Table 1 shows the demographics of the participants. Most of the participants in the two platforms sampled are Asian. This is consistent with earlier findings that the active online freelance community is predominantly Asian [6]. Participant selection was via convenience sampling [36], i.e., profiles that appeared on top of the search list based on our search criteria. We did not hand-pick freelancers by narrowing search criteria for particular continents or countries in order to achieve representative demographics.

3 Quantitative Data Analysis

This section presents the quantitative analysis: Section 3.1 describes scoring of participants' responses to code review tasks; 3.2 discusses inter-rater reliability; and 3.3 discusses the results.

3.1 Scoring Code Review Tasks

The open-ended responses required a systematic coding mechanism to identify evidence of developers' security engagement with code. For this we built a codebook that was used independently by three raters to identify evidence of participants' security orientation. The code book [5] was built upon the baseline established by existing well-known security resources (i.e., OWASP Top 10 [25] and CyBOK [26]) to classify vulnerabilities. We used the definition of security and its associated attributes from Avižienis et al.'s work [3]. OWASP and the CyBOK [26] both differentiate between identification and mitigation of vulnerabilities. Using this approach, we scored participants on a progressive scale of 0/1/1/2 for each code vulnerability: 0 if the particular vulnerability was neither identified nor mitigated; 1 if the particular was identified; 1 if the particular vulnerability was mitigated; and 2 if the particular vulnerability was both identified and mitigated.

We consider mitigation of a vulnerability with its explicit identification as a vulnerability versus mitigation of a vulnerability without identifying it as a vulnerability to be an important distinction, where the former provides evidence that the mitigation had a security rationale. First, this distinction is drawn from the resources built by the security community to understand implementation vulnerabilities. Research in developers' security also reports non-security rationales behind participants' secure choices; e.g., Van der Linden et al. [34] observed that many secure choices were made with non-security rationales, and Acar et al. [2] reported that developers who provided secure solutions did not believe that they had done so. Second, psychology literature also suggests the importance of understanding what people think they are doing, as the thinking behind action relates to the stability of actions over time, and understanding thinking has the potential to explain apparent inconsistencies in individuals' behavior [33]. We thus consider the distinction in secure choices (i.e., with or without security rationale) an important one, with the assumption that a recommendation with an explicit security rationale more strongly reflects a security orientation in developers' thinking.

We scored each participant for addressing any vulnerability that exists in the code-snippet. The code snippets have 2 seeded vulnerabilities, i.e., vulnerabilities planted in the code snippet by the source, and 6 additional security vulnerabilities, i.e., vulnerabilities picked inductively from participants' responses in the pilot study (Table 2). Notably, the full data set of 124 responses did not reveal any new vulnerabilities.

The total score of participant's security engagement with code is the sum of their scores on seeded vulnerabilities ($Total_{seeded-Vul}$) and on additional vulnerabilities ($Total_{additional-Vul.}$), i.e.,

$Total_{sec-eng} = Total_{seeded-Vul.} + Total_{additional-Vul.}$

With 8 vulnerabilities in the two code snippets, the total score for participant's security engagement with code was from 0 to 16.

A separate binary score, *security_awareness*, was also assigned to each participant: 1 if participant identified any vulnerability (whether correct or not), or talked about security without building on it; otherwise 0. This independent binary score notes whether participants think about security 'upfront' or not – irrespective of whether they identified vulnerabilities correctly or provided appropriate security solutions.

3.2 Inter-Rater Agreement

The open-ended answers to the code review tasks were scored independently by three raters. Two raters independently coded all the responses, and the third rater independently coded two different sub-sets of 10% of the responses from each experimental group.

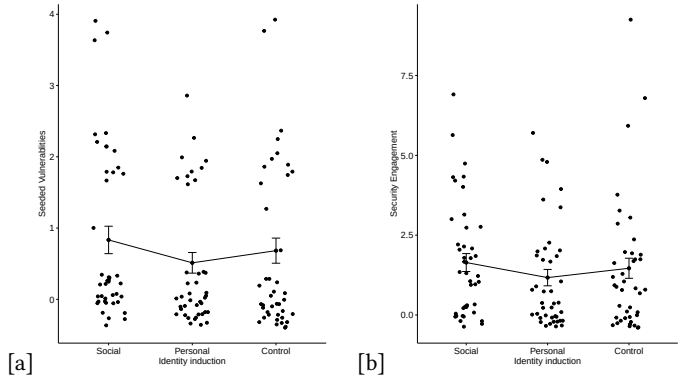
Two raters built the codebook together, so they had the same understanding of the vulnerabilities. As a result, IRR (inter-rate reliability) for 2 coders for all 124 responses was strong with Cohen's kappa values of 0.9 ($Total_{sec-eng}$) and 0.7 ($Total_{seeded-Vul.}$)

[4] https://stats.oarc.ucla.edu/other/gpower/

[5] The codebook and examples of how responses were scored are available in the online repository https://www.dropbox.com/sh/vrefbk2xshnwpmf/AADgS6iGFPq2GD2V-XmLpwBva?dl=0.

	Total	Geographical Disribution							Gender	
		Asia	M.East	Europe	Africa	S.America	N. America	Australia	Male	Female
Upwork.com	82	52%	15%	12%	9%	4%	7%	1%	81	1
freelancer.com	42	10%	10%	10%	5%	5%	0%	0%	41	1

Table 1: Participant Demographics by Geographical Distribution and Gender



[a] [b]

Figure 1: Distribution of scores on experimental conditions for (a) Seeded Vulnerabilities (b) Total Vulnerabilities

and 0.8 (*security awareness*). The IRR with the third rater on 10% data, who was not involved in developing codebook, was moderate to substantial using Fleiss's Kappa, i.e., 0.4 ($Total_{sec-eng}$) and 0.7 ($Total_{seeded-Vul.}$)and 0.79 (*security_awareness*). Upon discussion of the differences for $Total_{sec-eng}$, the third rater reported usability issues with the code-book. After the usability issues were addressed, the IRR remained the same on the new 10% data from each experimental group. A face-to-face (online) meeting of three raters concluded that, due to the open structure, some responses are ambiguous and can be coded differently.

3.3 Results

Figure 1 shows distributions of participants' security scores on both $Total_{seeded-Vul}$ (1a) and $Total_{sec-eng}$ (1b) with respect to experimental conditions. It is notable that attention to security is not prominent. Performing the preregistered analysis to compare social identity primes with a one-way ANOVA resulted in p-value=0.5, suggesting that there are no significant differences between groups. We further performed Tukey multiple pairwise-comparisons between the means of groups. The results show no significant differences between any pair (Personal-Control:p=0.8, Social-Control: 0.9, Social-Personal: p=0.5). Given the profile of low scores, we validated the parametric ANOVA results by also conducting post-hoc Kruskall-Wallis analysis [20] on: $Total_{secure-eng}(p = 0.23)$, $Total_{seeded-Vul.}$ (p=0.53), and *security_awareness* (p=0.23), which mirrored the above outcomes with no significant group differences.

Our design allowed us to run a manipulation check on the social identity induction, to confirm that participants identified with the intended identity groupings, as found with prior use of the three-things paradigm [27]. They also included four potential mediators that looked for possible influence of responsibility and reputation. That is, we checked how effectively the different identity groups were created. The Cronbach's alpha of the four social identification questions is 0.73, showing that they make a reliable scale, i.e., alpha > 0.7. However, the three conditions did not differ reliably (Social = 5.6 , Personal = 5.9 , Control = 5.9). Looking at the means of the four potential mediators individually for each of the conditions, none of the four potential mediators showed statistical significance for any of the questions in the three conditions, i.e., Social: 6.05, 6.45, 5.98, 6.36, Personal: 5.98, 6.61, 5.61, 6.27, Control: 5.98, 6.37, 6, 6.2. As such, the limited impact of different social identity groups in Figure 1 can be traced back to the lack of strong evidence that these groups were clearly created.

3.4 Answering RQ1:

Because the statistical analysis of data was inconclusive, we cannot reject our hypotheses with confidence. We obtained a floor effect in participants' responses in each experimental group. Manipulation checks also showed that the participants were not successfully primed for the identity conditions. Hence, we cannot conclude whether social priming had any effect on participants' security engagement with code. One post-hoc speculation for why the three things manipulation did not replicate the effectiveness of past studies is that, in the software development context, the personal and social identity contrast is more ephemeral or, at least, less salient to participants in the context of the study.

Despite the inconclusive results of psychological priming, the rich data collection enabled us to explore: *Do developers engage with security in code irrespective of the priming and control conditions?* Figures 2a and 2b show the frequency of participants' scores on $Total_{seeded-Vul}$ (Figure 2a) and $Total_{sec-eng}$ (Figure 2b), irrespective of the priming and control conditions. This shows that only a few participants identified more than one security issue. Fig. 2 (c) provides a comparison of the three variables - $Total_{seeded-Vul}$, $Total_{sec-eng}$ and $security\_awareness$. An interesting comparison is between the percentage of participants who scored above zero on $Total_{secure-eng}$ (top, Fig. 2 (c)) vs. $Total_{seeded-Vul.}$ (middle, Fig. 2 (c)). When evaluated only for seeded vulnerabilities, 31% scored above zero (of which only 29% suggested mitigation), whereas 56% of the participants scored above zero for all vulnerabilities. This shows the sensitivity of our scoring scale in measuring developers' security engagement with code. We note in Fig. 2 (c) (bottom) that 44% of participants talked explicitly about security when engaging with code without security priming.

Participants' familiarity with OWASP showed no significant relation to their security engagement with code (53% Yes (n=66), 47% No (n=58)). Some participants who said they are not aware of the OWASP list, mentioned vulnerabilities included in the list. Of those who said yes, 14% (n=9) appeared to have copied the OWASP vulnerability list from online resources. This signals developers' tendency to copy-paste from online resources, as reported in earlier

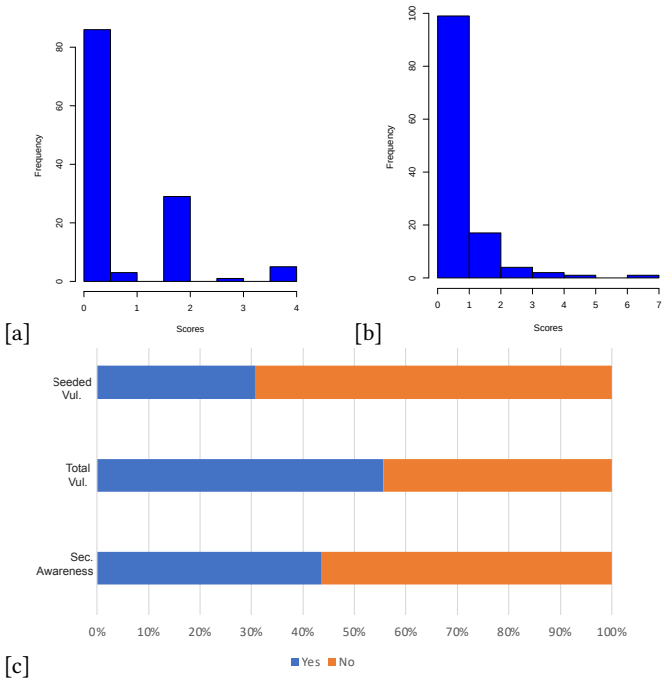


Figure 2: (a) Frequency of scores on Seeded Vulnerabilities (b) Frequency of scores on Total Vulnerabilities, (c) Positive security scores for security awareness (bottom) , total vulnerabilities (middle) and seeded vulnerabilities (top)

studies [1]. We also consider it noteworthy that, despite the best efforts of the security community to disseminate a list of common code vulnerabilities, many developers working on global projects are not familiar with the OWASP list.

More than half of the participants engaged securely with the code despite not being primed for security *per se*. It is noteworthy, as we are picking developers from a global marketplace whose security education, skills, and training are not known (in contrast to student samples and professional setups where a general understanding of their security education and security skills is generally known). These participants were not primed for security, yet there was indication of attention to security when engaging with code. Nevertheless, the security perspectives of developers vary.

4 Variations in Security Perspectives

We conducted an inductive analysis [32] of the 138 responses to the 2 code review tasks for 69 participants who scored positively for security engagement with the code. Each response contains answers to four questions, asked at the end of the code review tasks, that probe what participants think the code does, what they notice about the code, actions they would take to correct it, and why. We studied answers to these questions collectively. Participants' responses were read several times and were discussed in detail by the two researchers who coded the data in full.

Three broad themes were identified in the analysis: how security issues are identified (4.1), how security issues are mitigated (4.2) and how participants reason about the security issues in the code snippets (4.3). These and their sub-themes are discussed in the sections that follow. The overarching message emerging from the themes was that developers vary in their security frames of references and deal with security with different approaches and for different reasons. Table 2 shows how these security issues were identified, mitigated, and reasoned about by the participants. We combined the last three vulnerabilities (insecure communication channel, insecure payment processing, and unencrypted sensitive data) under "security issues with payment processing", as raters agreed that all three belonged to insecure payment processing.

4.1 Identifying Security Issues

Here, we explore *how* participants identified the vulnerabilities.

t4.1.1 Types of Vulnerability Identified: Participants identified eight vulnerabilities in the code. Participants used more than one security technique to identify some vulnerabilities. Table 2 shows the terms participants used to talk about these vulnerabilities. For vulnerability 1, *insecure password storage*, participants mentioned that the password is being saved in plain text, not encrypted nor hashed. For vulnerability 2, *check for weak password*, participants noticed missing checks for password length and combination of characters. For vulnerability 6, *security issues with payment processing*, participants identified insecure payment processing, using unencrypted payment data, and use of insecure communication channels. For the other vulnerabilities, *retrieving data of all users*, *unique username check*, and *logging sensitive information*, participants identified one particular aspect of code that can lead to vulnerability.

t4.1.2 Inconsistent Use of Security Terms: The way in which participants talked about vulnerabilities suggests that developers often use security terms inconsistently from one other. In some cases they appear to understand the concept but apply the wrong term, and in others they have confused the concepts. For example, OWASP strongly recommends hashing of passwords [25] for storage and not to use encryption. As an example of confounding concepts, UP12 stated that passwords should be saved with encryption and suggested using Django for password storage because it "..ensures the password data entered is irreversible". However this is a characteristic of hashing, not encryption. As an example of applying incorrect terms, FP10 and UP15 both noticed that the password was not encrypted and suggested that the password should be hashed. This illustrates that some developers mix different secure (i.e. hashing) and insecure (i.e. encryption) terms in a given scenario, even though they appear to understand the underlying concepts. Some developers, however, demonstrated command over both security language and security knowledge, e.g., UC19 stated explicitly: "Passwords should never be stored in plain text, should also not be encrypted using an encryption algorithm; it should be hashed so no one can decrypt the password". UP12, FP10, and UP15's use of inconsistent language may indicate that they have insufficient security education. However, taken together these cases (plus UC19) suggest that inconsistent use of terms is common and may lead to a false perception of developers' security knowledge.

The data also shows how developers often use the terms *confidentiality* and *privacy* loosely. Confidentiality is one of the attributes of security that requires "absence of unauthorized disclosure of information" (p.95, [3]); *privacy* relates to an individual's right over

1.Insecure Password Storage				
Id.	plain-text (e.g., UC19, US10)	not encrypted (e.g., UP15)	not hashed(e.g., UP15)	
Mit.	encryption (e.g., FS11)	hashing (e.g.,FP10)	hashing with salt (e.g.,UP15)	Built-in models (e.g.,UC21)
2.Check for weak password				
Id.	password length (e.g.,FP13,FP3)	combination of characters (e.g., FP1,FC4)		
Mit.	check for: min. length (e.g., FP13) , max. length (e.g., FC4, UC24)	Combination of characters(e.g., FP1,FC4)		
3. Retrieving data of all users				
Id.	All users' data is retrieved (e.g., FS7, FS6)			
Mit.	remove it			
	availability issues (e.g.,FS6)	privacy issues (e.g., US5)	confidentiality issues (e.g.,FS7)	unnecessary code (e.g. FP13)
	reduce network data (e.g., UC5)	improve efficiency (e.g., FS6)	scalability issues (e.g. UC5)	
4. Unique Username check:				
Id.	missing check for unique username (e.g., FP1, FS3, FS6)			
Mit.	add a check for unique username			
	security concerns (e.g., UP15)	avoid confusion (e.g., UC7)	code maintainability (e.g., FC3)	avoid errors (e.g., FP1)
5. Logging Sensitive information				
Id.	User's sensitive information is logged (e.g., FC3, FC6, FS10)			
Mit.	remove it (e.g., FS10)	hash it (e.g., FC3)	encrypt it (e.g., UC5)	
6. Security Issues with payment processing				
Id.	insecure processing ((e.g, FP10)	unecrypted data (e.g, US18)	insecure comm. channel (e.g., UP12)	
Mit.	use secure channel (e.g., UP12)	encrypt users' info (e.g., FP10)	use 3rd party payment service (e.g., UC12)	

Table 2: Theme 1: Security Issues identified (Id.) and mitigated (Mit.) by Study Participants. Last 3 vulnerabilities are combined. Blue cells denote reasoning that is not about security.

individual's identifiable information [30]. Researchers also at times tend to conflate the two terms (e.g.,[4]), but privacy and confidentiality are two different concepts. Some participants talked about users' "privacy" concerns if all information is retrieved from the database and sent to the register page of a new user, while another noticed that all existing users will be "exposed to the unauthorized user". Similarly, while some participants identified "privacy" concerns in logging users' credit card information, others noticed "confidentiality" issues with logging users' credit card information.

4.2 Mitigating Security Issues

Participants varied in how they mitigated security issues in code.

t4.2.1 Variation in mitigating Specific Vulnerabilities: Table 2 shows how participants varied in their approaches to mitigating the same security issues, though not all of them are correct. For example, checking for minimum length of password and combination of characters is a correct approach to ensure strong passwords, but checking for maximum length is not[6].

Similarly, removing a user's payment information from logging on the server is appropriate, rather than encrypting it or hashing it. For other vulnerabilities, participants suggested mitigation approaches that are correct. However, as vulnerabilities *security issues in payment processing* and *check for weak passwords* require more than one code-fix, most of the participants did not suggest all of them. We see these variations as an indication that, even when developers attend to security, they may not consider whether their actions are complete and sufficient.

For example, to mitigate a missing check for weak passwords, some participants suggested a limit to maximum password length, a measure which NIST guidelines warn against. Although these participants suggested other password security measures (e.g., FC4 suggested using a combination of characters for a strong password), incomplete security measures, such as limiting maximum password length, can make the code vulnerable.

t4.2.2 Types of Suggested Solutions: Among the different types of suggested solutions, participants often suggested using third-party technical solutions. Although using third-party payment processing and authentication services is recommended by the security experts [7], many developers opted to use a third-party solution for reasons other than security. Table 3 shows how participants used third-party technical solutions to address different issues in code, suggesting that 64% of participants used them for non-security reasons. Three different types of third-party solutions were suggested by the participants: *Django's built-in packages*, *payment processing services*, and *authentication services*.

Participants who suggested using Django's built-in features or other third-party services, instead of implementing their own, mostly preferred them for non-security reasons. Reasons included improved efficiency, code maintainability, powerful processes, easy to use, and social acceptance. We reflect that security in code is an 'intangible characteristic', that may not be readily evident to developers; hence, it needs to be 'sold' to developers as something that provides immediate benefits that they can observe.

4.3 Types of Security Reasoning

This section unpacks how participants varied in their reasoning for suggesting secure actions.

t4.3.1 Social Considerations: We noted mentions of all social considerations in participants' responses and observed that most of the social considerations led to secure actions. Table 4 shows which

[6]https://jumpcloud.com/blog/review-of-nist-sp-800-63-

[7]https://www.securecodewarrior.com/

different types of social considerations led to which code quality. The blue cells show non-security reasoning for participants' action, which are few. Participants mentioned concerns about users suffering loss from information leaks and to protect users from making mistakes by suggesting stronger passwords [US18, UC24]. Some participants (tended to) perform secure actions on ethical grounds [e.g., UC14], while others were fearful of audit companies not approving their applications [e.g., UC12] or of malicious insiders and outsiders exploiting insecurely-saved data on servers [e.g., UC24, UP21]. Participants were also motivated to ensure their secure practices in order to maintain users' and customers' trust [e.g., UP12, FP10]. We also saw evidence of participants' excessive trust in 'big players'; for example, UC24 suggested logging users' card information only if servers are as secure as those of Visa/ MasterCard. This echoes earlier findings that suggest that developers tend to trust 'big players' [19]. Nevertheless, while trusting and using prepared solutions by 'big players' is generally encouraged, it is important that developers have critical security knowledge to make informed decisions. For example, in this case, logging of users' card information on servers is not secure even for 'big players', which UC24 thought is not harmful. Additionally, participants were concerned about following social norms surrounding security practices. While some talked about recommended ways and best practices when using Django built-in capabilities [UC22, FC10], others talked about best security practices for fear of being labelled newcomers by the developer community [UC3].

t4.3.2 Implementation Vs. Design Vulnerabilities: We noticed variation in how the participants looked at design vulnerabilities, i.e., problems that arise due to software design issues vs. implementation vulnerabilities, i.e., problems that arise due to issues in code that is working functionally well but has security problems. We presented the code snippets to participants expecting them to notice implementation vulnerabilities, but we realized that developers engage with code at the design level simultaneously. While the technical vulnerabilities had clear security rationale, participants approached design vulnerabilities differently, as shown in Table 2. Vulnerabilities 2 and 3 are design-level vulnerabilities, and participants varied in their reasoning on why they should be fixed, oscillating between security and non-security reasons. While some noticed *retrieving information of all users* because of availability issues [e.g., US1, FS6], compromise on users' privacy [e.g., FP13], and confidentiality [e.g., US25, FS7], others suggested removing it for improving efficiency [e.g., US21, US17], getting rid of redundant code [e.g., FP13, US9], and scalability concerns [e.g., UC5].

Similarly, while many participants noticed *check for unique username* in code and suggested fixing it for security reasons [e.g., UP15], some other participants suggested removing it to avoid confusion [e.g., FS13, UC7], for code maintainability [e.g., FC3], and to keep the application functional [e.g., FS6, FP1].

4.4 Answering RQ2

The pre-registration plan was to relate qualitative analysis to quantitative findings. However, as the findings with respect to RQ1 were inconclusive, we present the findings with respect to RQ2 without drawing such a relation.

The qualitative analysis of participants' responses suggested that, when security requirements are not defined explicitly in specification documents, participants approach security in code based on their own perceptions of security, rather than what software owners (or researchers for that matter) expect them to address. Participants used varying language to identify security vulnerabilities and sometimes mitigated vulnerabilities differently from one other. The reasons for attending to security also varied. Participants approached security in their own context, and developers' security is a context-sensitive problem. Evidence suggests that use of third-party and built-in technical solutions is often adopted as part of developers' preferred way of working due to different non-security reasons related to developers' immediate contexts. Additionally, social considerations for other developers and for the wider good, were often a key motivation for participants' security actions.

5 Discussion and Future Work

This analysis suggests the need for a holistic look at developers' secure coding behavior, with carefully-crafted empirical studies which recognise developers' complex socio-technical contexts.

Socio-technical systems envision a greater involvement of human action in engineering solutions [24]. Socio-technical researchers highlight that "software design methods are geared towards developing a solution to 'the problem"', thus if that 'problem' is not understood properly then applying the methods will generate an "inappropriate solution." (p.13,[5]). In order to address appropriately the problems that developers face in writing secure code, it is important to understand what security means in their context. Providing security interventions in the context in which developers sit is recognized in existing literature [10].

5.1 Developers think about security differently

Developers have often shared blame for not thinking about security [23] [37]. Wurster and Oorschot [37] considered it unrealistic to expect secure code from developers in general, as they vary in their skills, of which security is often not a part. Naiakshina et al. conducted a field study with freelance developers and suggested that the "majority of non-prompted freelancers did not think about security" (p.2.[23]). This work explores whether developers think about security at all, irrespective of whether they provide correct security solutions. It shows that, without prompting for security, the majority of developers (56% of participants) engaged securely with code, with nearly half of the participants (44%) being explicitly conscious about security. However, developers varied in how they noticed security in code, how they talked about it, and how they dealt with it. We scored participants' security behavior based on *any* security vulnerabilities they noticed which are accepted in the security community as a vulnerability, while previous studies scored participants only on seeded vulnerabilities, i.e., vulnerabilities researchers expected participants to notice and address with correct security solutions. The work of Lopez et al. [19] on understanding how developers talk about security with one another analysed developers' conversations in Stack Overflow and observed that developers are interested in security and engage in meaningful conversations about security. While the work of Lopez et al. [19] focused on how developers talk about security with one another, we capture developers' engagement with security when they engage

1. Django's Built-in Packages	
Improve code	maintainability (cleaner code,less code, avoid code repetition), readability, flexibility (e.g., US28,US26,UC22)
Improve Efficiency	avoid extra query to database, save time (e.g., UC1, UP12)
Make things easier	easy to test, easy to handle database, easy to do desired task, avoid uncertain errors (e.g., UP21, UP19)
Powerful processes	lots of features, comes with default models, automates tasks, can build complex projects (e.g., FC10, UC14, UC13)
Trust	rely on proof-tested system, avoid rewriting, build on years of programming (e.g., FC10)
Social Acceptance	recommended way to use, part of best practices, avoid being labelled a newcomer (e.g., UC22, FC10)
Security	avoid security holes, automatic hashing of passwords, make applications more secure (e.g., UP12, FC10, US28)
2. Payment Processing Services	
Other reasons	save time, write tests more easily (e.g., UC10, UC13)
Security reasons	conform to PCI standards, security of customers' data, avoid threat to business security (e.g., UC12, UC5)
3. Authentication Services	
Other reasons	smooth registration service, powerful and easy (e.g., FC10)
Security Reason	not storing data on server, protecting app from attacks (e.g., UC10, UP19)

Table 3: Theme 2 - Participants' Use of Other Technologies - Blue cells denote non-secure reasoning. Blue cells denote reasoning that is not about security

Social Considerations			
1. Care and concern for users	ensure users' data confidentiality (US25)	ensure users' privacy (UC5)	avoid mistakes by users
2. Ethical responsibility	remove logging of sensitive data (UC14, US21)		
3. Fear of security watchdogs	remove logging of sensitive data (US21)	ensure secure payment processing (UC12)	
4. Fear of malicious insiders and outsiders	ensure secure password storage (e.g., UC24)	ensure secure payment processing (e.g., UP21)	
	remove logging of sensitive data (e.g., UC24)		
5. Trust by / in others	ensure secure password storage ((e.g., FP10)	ensure secure payment processing (e.g., UP12,)	remove logging of sensitive data (e.g., UC24)
6. Social norms	avoid security holes in user registration (UC3)	remove logging of sensitive data (UP26)	use Django's built-in features (UC22)
	get better code maintainability (FC10)	get scalable application (FC10)	

Table 4: Theme 3 - Social Considerations and their goals in code – Blue cells denote non-secure reasoning

with code through reflection and review. Our observations on the importance of context are also in sync with research in behavioral science. Building on action identification theory [33], Hunt and Hoyer [16] observe that when researchers assume actions as objects and researchers' identification of action as "correct", without measuring how the behavior is conceptualized by individuals, they face the risk of losing important information, and this affects the maintenance of action as well as the emergence of new behavior. They consider this a potential bias. We thus consider the sensitivity of our scoring approach in measuring developers' engagement with security during code review to be important, i.e. to quantify developers' security based on how they perceive it.

5.2 Leveraging the socio-technical context

This section considers how socio-technical context can be leveraged to improve secure code development.

t5.2.1 Diversity of developers' perspectives should be accommodated: Developers' diversity, with varying skills and experiences, includes different viewpoints on security for the same piece of code. While initially we expected developers to find one seeded vulnerability in each task, after the pilot studies we scored participants for eight vulnerabilities for both the tasks. This illustrated that "more eyes see more", with the different security viewpoints combining into a richer overview. This suggests that researchers and toolsmiths should engage with the diversity of developers' viewpoints on code security and seek to accommodate and potentially harness it, rather than considering developers as 'outside the problem space'. Security tools should aim to support them in their diverse contexts, facilitating them to consider security more broadly. The problem with many existing (security) interventions (e.g., automated security tools) is their technical focus, excluding the non-technical aspects of the problem [28]. Unless security interventions are contextualized to developers' needs, security interventions will find little space in developers' routine tasks.

t5.2.2 Security should add value to a developer's immediate context: Developers feel comfortable adopting third-party services and built-in packages of the development frameworks. Security experts encourage developers to use these 'prepared' solutions instead of writing their own. However, developers are eager to adopt these 'prepared' solutions for many non-security reasons such as convenience of use, efficiency of solution, and code maintainability. We suggest that, while security may not yet be a *tangible* quality to developers, it needs to be packaged with other code quality attributes that are visible to developers in their immediate contexts.

5.3 Challenges for multidisciplinary studies

Investigating developers' security practices in real contexts is difficult, because factors that influence developers' thinking are hard to control and account for. We conducted a multidisciplinary study to understand whether priming developers to think of themselves in

terms of a personal or social identity influenced their secure coding behavior; unfortunately, the study found no significant differences between experimental conditions.

The design of the study prompted individuals to think of themselves either as member of the developer community or as an individual or without such priming, in order to observe the effect of priming on their security engagement with code. We made careful efforts to recruit participants without priming them to think of themselves as developers – in that we faced many challenges. We had to give up one recruiting path[8] that required screening participants as a "developer". We recruited developers from an open market of freelancers without explicitly mentioning the word 'developer' in its job posting. As the job posting mentioned 'code-review tasks', the risk of being primed as developers existed. To address this, we framed psychological priming questions by mentioning that the study is in collaboration with psychologists to study some human factors in coding and to help the psychologists know more about them as individuals – followed by priming questions. The psychologist co-authors considered this sufficient to nullify any effect of priming during recruitment. However, the priming results showed that participants were not successfully primed in different conditions. The closer look at the responses showed that the majority of participants in each of the conditions talked about themselves as developers; hence, our understanding is that they came to the study thinking of themselves as developers, and that this inherent identity may have limited the effect of the priming.

Although all participants were paid £10/ hour (net amount) as an incentive to participate in the study, we realized that participants were implicitly incentivised to improve their developer profile on the freelancing platform. Many freelancers contacted us after the study to give them feedback and mark the job done. It improved their rating, pushed them up in search algorithm, eventually helping them get more clients. This possibly explains further why the social and personal identity priming was not successful. Nevertheless, the rich data helped us make meaningful assessment of how developers engage with code, even though priming was not successful. The detailed report on challenges we faced in recruiting participants for this multidisciplinary study can be found here [29].

5.4 Issues with Security Knowledge

t5.4.1 Inconsistent Security Terminologies: The inconsistent use of terminology is identified as one of the challenges to advancing research in usability, security, and privacy in the 2010 report on the National Academy of Sciences Workshop [9]. Similarly, this study provides evidence that developers from industry use the terms *confidentiality* and *privacy* loosely – sometimes with the same implications. Additionally, while encryption and hashing are different ways to secure data for different purpose, developers often used these words interchangeably. Research needs to unpack such subtle differences in developers' understanding of security terms and assess developers' security in the context of the security cultures to which they belong. Practitioners need to devise interventions that propagate consistent use of security terminologies among developers, addressing shortcomings in developers' security understanding and miscommunications that may arise among stakeholders due to the inconsistent use of terminologies.

t5.4.2 Lack of Familiarity with OWASP Top 10 Vulnerabilities: The OWASP list of Top 10 vulnerabilities is one of the most-disseminated security awareness resources by the security community. Hence, as security researchers and practitioners, we considered knowledge of the OWASP vulnerabilities as a reasonable yardstick to gauge developers' familiarity with security knowledge. The responses of the participants, however, showed that many participants were either totally unaware of the OWASP list [e.g, FP1, UC22] or had knowledge of vulnerabilities but did not know them as part of the OWASP list [e.g., UP12, FC14]. We suggest that security researchers and industrialists need to disseminate security information more widely, with an understanding of the diversity of the software development community. This includes, for example, developers who may work outside of organisational boundaries and work in different socio-cultural contexts. This also requires an understanding of the diverse communication channels such as discussion forums, software freelancing platforms and code repositories, etc.).

t5.4.3 Incomplete Knowledge of Password Protection:

Weak password checks is one of the top OWASP vulnerabilities and, despite implementing secure password storage, information of users with weak passwords can be hacked easily via sophisticated technologies. Developers' security engagement when working with passwords is often assessed in terms of how developers store passwords, for example in the work of Naiakshina et al. [22] and Hallet et al. [14]. Implementation of strong password checks is also a recommended OWASP practice which is often overlooked. We observe that many developers who suggested secure storage of passwords, failed to notice that strong password checks were not implemented. On the other hand, others noticed missing checks for strong passwords, but failed to store passwords securely. In some cases [FC4 and UC24] developers even suggested insecure ways to validate passwords to suit their personal preferences, such as reducing the maximum character length of passwords. This example highlights the need to investigate how holistic developers' security knowledge and thinking is, and how to promote more holistic reasoning.

6 Limitations of the Study

We studied participants' security engagement with code irrespective of identity priming in this work. Although participants were not primed to think about security, there could be a chance that participants' coding behavior may have been influenced by identity priming done in the beginning of the study. To assess whether any such effect was present, we studied the responses to the social identification check questions and the open answers to the identity priming questions. We found no evidence of successful identity priming. Hence, we can safely say that participants were not primed by the design of our study.

Due to the unknown socio-technical contexts of participants in the freelance community, we cannot say whether the difference between participants who engage with security in code and those who do not is due to their previous exposure to security knowledge, security skills, and security culture. We tried to address this limitation by addressing a security-specific question about a well-known security source, i.e., OWASP, to capture how well-acquainted they

[8] Qualtrics Research Services

are with common security knowledge. We did not find any correlation between participants' answers to this question and their security engagement with code. However, we do accept that our sample might be biased towards developers who may not have technical security skills or biased towards a certain region. However, our sample is representative of general freelance community that is hired online. Additionally, since our sample also includes participants from other regions, we performed a sanity check, comparing data from different regions, but did not find any noteworthy differences. Our work was designed to seek evidence of security in developers' thinking when they engage with code irrespective of their security knowledge and experience. This preliminary study provides evidence of security in participants' thinking. We plan to control for other factors in future studies.

7 Conclusion

This paper reports a psychologically-informed study to understand how socio-technical contexts affect the ways developers attend to and engage with security in code. The paper presents a multi-faceted account of the observed behaviours, based on quantitative and qualitative assessments of participants' open responses to code review tasks, and finding that developers' approaches to security vary. The participants varied in how they noticed security issues in code, how they addressed them, and why they chose particular code changes. This evidence suggests that security occupies a complex decision space, and that many developers do think about security – but that they think about it differently from one another, bringing different viewpoints to security in code.

In order to embrace this diversity in developers' security frames of reference, more empirical studies need to consider developers' socio-technical contexts and take note of how developers identify with their actions, which can have implications for maintenance of their actions [33], [35]. It further suggests the need for innovation in addressing security bottom-up, with a more collaborative and contextualised understanding of developers' perceptions of security. We conclude that, only once we are able to comprehend how security is regarded by developers, can we provide interventions that are appropriate to the individual development context in order to influence their secure coding behaviour.

8 Acknowledgements

We thank all the study participants and acknowledge Joseph Hallett for his input as third rater for scoring participants' responses. The work was partially supported by UKRI/EPSRC (EP/P011799/1, EP/P011799/2, EP/R013144/1 and EP/T017465/1), NCSC, and SFI (13/RC/2094 and 16/RC/3918).

References

[1] Y. Acar et al. 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, 289–305.

[2] Y. Acar et al. 2017. Security developer studies with github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)*. 81–95.

[3] A. Avižienis et al. 2004. Dependability and its threats: a taxonomy. In *Building the Information Society*. Springer, 91–120.

[4] R. Balebako et al. 2014. The privacy and security behaviors of smartphone app developers. (2014).

[5] G. Baxter and I. Sommerville. 2011. Socio-technical systems: From design methods to systems engineering. *Interacting with computers* 23, 1 (2011), 4–17.

[6] N. Beerepoot and B. Lambregts. 2015. Competition in online job marketplaces: towards a global labour market for outsourcing services? *Global Networks* 15, 2 (2015), 236–255.

[7] L. Braz et al. 2021. Why Don't Developers Detect Improper Input Validation?'; DROP TABLE Papers;–. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 499–511.

[8] Top Coder. 2021. On-Demand Talent Trends Report. Retrieved June 18, 2021 from https://www.topcoder.com/blog/talent-trends-report-part1/

[9] National Research Council. 2010. *Toward Better Usability, Security, and Privacy of Information Technology: Report of a Workshop*. National Academies Press.

[10] A. Danilova et al. 2020. One size does not fit all: a grounded theory and online survey study of developer preferences for security warning types. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*. IEEE, 136–148.

[11] A. Danilova et al. 2021. Code Reviewing as Methodology for Online Security Studies with Developers–A Case Study with Freelancers on Password Storage. In *Seventeenth Symposium on Usable Privacy and Security ({SOUPS} 2021)*. 397–416.

[12] B. Doosje et al. 1995. *Perceived intragroup variability as a function of group status and identification*. Vol. 31. Elsevier. 410–436 pages.

[13] B. Doosje et al. 1998. Guilty by association: When one's group has a negative history. *Journal of personality and social psychology* 75, 4 (1998), 872.

[14] J. Hallett et al. 2021. "Do this! Do that!, And Nothing will happen": Do specifications lead to securely stored passwords?. In *43rd International Conference on Software Engineering* (43 ed.). Institute of Electrical and Electronics Engineers (IEEE), United States.

[15] S Alexander Haslam. 2004. *Psychology in organizations*. Sage.

[16] G. W Hunt and W. D. Hoyer. 1993. Action identification theory: An examination of consumers' behavioral representations. *ACR North American Advances* (1993).

[17] O. Kononenko et al. 2016. Code review quality: How developers see it. In *Proceedings of 38th international conference on software engineering*. 1028–1038.

[18] M. Levine et al. 2005. Identity and emergency intervention: How social group membership and inclusiveness of group boundaries shape helping behavior. *Personality and social psychology bulletin* 31, 4 (2005), 443–453.

[19] T. Lopez et al. 2018. An investigation of security conversations in stack overflow: Perceptions of security and community involvement. In *Proceedings of the 1st International Workshop on Security Awareness from Design to Deployment*. 26–32.

[20] P. E McKight and J. Najab. 2010. Kruskal-wallis test. *The corsini encyclopedia of psychology* (2010), 1–1.

[21] D. Mendez et al. 2020. Open science in software engineering. In *Contemporary Empirical Methods in Software Engineering*. Springer, 477–501.

[22] A. Naiakshina et al. 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*. 311–328.

[23] A. Naiakshina et al. 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*. 1–12.

[24] M. Ottens et al. 2006. Modelling infrastructures as socio-technical systems. *International journal of critical infrastructures* 2, 2-3 (2006), 133–145.

[25] OWASP Foundation, the Open Source Foundation for Application Security. [n.d.]. https://owasp.org/. (Accessed on 03/06/2020).

[26] Frank Piessens. 2019. *The Cyber Security Body of Knowledge*. University of Bristol, Chapter Software Security. https://www.cybok.org/ Version 1.0.

[27] T. Postmes et al. 2013. A single-item measure of social identification: Reliability, validity, and utility. *British journal of social psychology* 52, 4 (2013), 597–617.

[28] I. Rauf et al. 2021. The Case for Adaptive Security Interventions. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2021).

[29] I Rauf et al. 2022. Challenges of Recruiting Developers in Multidisciplinary Studies. In *Accepted for publication in 1st International Workshop on Recruiting Participants for Empirical Software Engineering (RoPES'22*.

[30] P. M Schwartz and D. J Solove. 2011. The PII problem: Privacy and a new concept of personally identifiable information. *NYUL rev.* 86 (2011), 1814.

[31] MA. Storey et al. 2020. *The who, what, how of software engineering research: a socio-technical framework*. Vol. 25. Springer. 4097–4129 pages.

[32] D. R Thomas. 2006. A general inductive approach for analyzing qualitative evaluation data. *American journal of evaluation* 27, 2 (2006), 237–246.

[33] R. Vallacher and D. M Wegner. 1987. What do people think they're doing? Action identification and human behavior. *Psychological review* 94, 1 (1987), 3.

[34] D. van der Linden et al. 2020. Schrödinger's security: opening the box on app developers' security rationale. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 149–160.

[35] D. M. Wegner et al. 1984. The emergence of action. *Journal of Personality and Social Psychology* 46, 2 (1984), 269.

[36] C. Wohlin et al. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

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

[38] A. Yamashita and L. Moonen. 2013. Surveying developer knowledge and interest in code smells through online freelance marketplaces. In *2nd International Workshop on User Evaluations for Software Engineering Researchers*. IEEE, 5–8.