A Survey on Developer-Centred Security

Citation for published version:
Tahaei, M & Vaniea, K 2019, A Survey on Developer-Centred Security. in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW).* Institute of Electrical and Electronics Engineers (IEEE), Stockholm, Sweden, pp. 129-138, 4th European Workshop on Usable Security, Stockholm, Sweden, 20/06/19. https://doi.org/10.1109/EuroSPW.2019.00021

Digital Object Identifier (DOI):
10.1109/EuroSPW.2019.00021

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)

A Survey on Developer-Centred Security

Mohammad Tahaei*, Kami Vaniea†
School of Informatics, University of Edinburgh
Edinburgh, UK
*mohammad.tahaei@ed.ac.uk, †kvaniea@inf.ed.ac.uk

***Abstract*—Software developers are key players in the security ecosystem as they produce code that runs on millions of devices. Yet we continue to see insecure code being developed and deployed on a regular basis despite the existence of support infrastructures, tools, and research into common errors. This work provides a systematised overview of the relatively new field of Developer-Centred Security which aims to understand the context in which developers produce security-relevant code as well as provide tools and processes that that better support both developers and secure code production. We report here on a systematic literature review of 49 publications on security studies with software developer participants. We provide an overview of both the types of methodologies currently being used as well as the current research in the area. Finally, we also provide recommendations for future work in Developer-Centred Security.**

***Index Terms*—Usable Security and Privacy, Developers, Software Development, Human Factors, Human Computer Interaction, Computer Security, Systematic Literature Review, Survey**

I. Introduction

Software is increasingly being integrated into all aspects of society, it controls everything from small home appliances such as kettles [1], to large systems like power plants [2], as well as data management infrastructures such as health records [3]. Each of these systems has a specific non-security goal (tea, power, health) but they also have an expectation from the public that they will provide other non-functional requirements such as safety, reliability, and privacy.

While it would be wonderful if developers could do all of that, instead we see that the introduction of security vulnerabilities into code is becoming a very large problem. The most common types of coding errors have remained relatively stable over time. Code Injection, for example, has topped the OWASP top ten vulnerabilities list for the last eight years [4] and 78.5% of recently scanned applications still suffer from it [5]. In 2013 alone, 88% of apps (out of 11,748) on Google Play had at least one cryptographic API mistake [6].

Developers, much like end users [7], [8], need support to create applications that are functional, usable, efficient, maintainable, and secure. The emerging and rapidly expanding area of Developer-Centred Security (DCS) aims to address some of these needs by applying existing methodologies from Human Computer Interaction (HCI) to the area of software development and security [9]–[12]. The application of HCI to software development has seen great success in the fields of API usability [13] and end-user software engineering [14].

To help developers make better use of security technologies many different approaches have been suggested such as security APIs [15], static [16] and dynamic [17] code analysis tools, and code creation processes such as pair programming [18]. However, many of these proposed solutions have had little success, potentially due to usability and workflow related issues. To systematically address these issues researchers need to understand the landscape of software development as it relates to both software developers and to security.

We present a formal structured literature review which identifies works that feature the trio of *security*, *software development*, and at least one study involving *users*. We identified 49 relevant research papers which we then sorted into eight themes: organisations and context, structuring software development, privacy and data, third party updates, security tool adoption, application programming interfaces, programming languages, and testing assumptions.

Our review highlights a lack of research in several aspects of DCS including, how to make security a business value and security often being ignored because it is a secondary requirement. To our surprise, even though programming languages are a fundamental tool in software engineering, only one paper discussed issues around security evaluation of programming languages with user studies. Software updates, which are critical for software security and a point of discussion in end-user usable security [19], are rarely discussed in DCS literature; with only one reviewed paper considering the challenges of using packages and libraries.

II. Systematisation Approach

We used a Systematic Literature Review [20] approach in order to identify all relevant literature. Two authors were engaged in every step, and decisions were agreed by both researchers to minimise the effects of bias and priming.

a) Selecting Literature: DCS is a relatively new area which crosses several fields resulting in papers in a range of publication venues. We decided to cover the top five publications in three fields: HCI, Software Systems, and Computer Security & Cryptography. To select specific high-quality publication venues in those areas, we used a Google Scholar feature that ranks scientific publications based on their h5-index and h5-median. We chose the top five listed venues (Table I). We also explicitly added the Privacy Enhancing Technologies Symposium, Symposium on Usable Privacy and Security (SOUPS), and IEEE Secure Development Conference. The first covers privacy areas, the second specifically

TABLE I
PUBLICATION VENUES REVIEWED PAPERS DRAWN FROM.

Publication	h5-index	h5-median	Total	Selected
Computer Security & Cryptography				
Computer and Communications Security	77	128	395	4
Security and Privacy	74	129	69	2
Information Forensics and Security	73	103	261	0
USENIX Security Symposium	70	106	66	1
International Cryptology Conference	62	84	65	0
Human Computer Interaction				
Computer Human Interaction	86	117	150	2
Computer-Supported Cooperative Work	56	79	34	2
Pervasive and Ubiquitous Computing	52	76	65	0
User Interface Software and Technology	45	72	17	0
Affective Computing	39	54	2	0
Software Systems				
Software Engineering	74	102	154	6
Software Engineering	56	83	79	1
Journal of Systems and Software	51	71	92	0
Foundations of Software Engineering	50	82	44	3
Programming Language Design and Implementation	50	78	30	0
Others				
Symposium on Usable Privacy and Security	31	60	265[a]	10
Secure Development Conference	NA	NA	15	0
Privacy Enhancing Technologies Symposium	NA	NA	119	0
Snowballing	NA	NA	21	18[b]
Final set			1943	49

[a] Has results from multiple sources.
[b] All items are selected, three items fit into one of the lists above.

targets usable security research, and the third is a newly established conference on secure development.

For the search itself, we used the The ACM Digital Library for ACM publications (SOUPS 2005 – 2013), IEEE Xplore digital library for IEEE publications, Engineering Village (EV) for PETS, International Cryptology Conference (CTYPTO), USENIX Security, Journal of Systems and Software, and SOUPS (2014 onward). We used EV because USENIX Security, CRYPTO, and ScienceDirect indexing websites do not support complex search queries. We limited our search to title, abstract and keywords. We also limited results to those published before October 13, 2018.

```
("security" OR "privacy" OR "cryptography")
AND
("human" OR "empirical" OR "user" OR "users" OR "interview" OR "interviews"
OR "survey" OR "surveys" OR "lab study" OR "laboratory study" OR "think aloud"
OR "cognitive walkthrough" OR "questionnaire" OR "questionnaires"
OR "usability" OR "usable")
AND
("developer" OR "developers" OR "development" OR "software" OR "app"
OR "application" OR "programmer" OR "programmers" OR "software engineer"
OR "software engineers" OR "system administrator" OR "system administrators")
```

Listing 1: Executed query

b) Practical Screen: The query in Listing 1 was executed on the 18 venues (Table I) which generated 1922 results. The resulting publications were then loaded into a reference management software (*Zotero*) which was used to remove duplicate papers as well as store notes taken during screening.

One researcher then went through and applied the following screening criteria by looking at the title and abstract of each publication. The researcher intentionally took a slightly broad view of the criteria in the first pass erring on the side of inclusion rather than exclusion.

Security - The paper had to directly involve cyber security, though it did not have to be the primary topic.

Software Development - The paper had to involve the process of software development or at least code creation. Management practices that directly impacted developers were also included.

User Study - The paper had to include a user study involving research subjects. Studies that only had artefact analysis, such as only looking at code samples, or tools were excluded.

Full Papers - Posters and extended abstracts were excluded.

The first pass resulted in 46 publications marked for potential inclusion. A second researcher then went through and reviewed all the included publications also looking through the publication content, they identified publications which did not meet the criteria, these were then discussed and 18 were excluded. The final set included 28 papers.

Snowball: To further improve our coverage, we also use a snowball approach to find additional literature. For each of the identified papers above, a researcher read through the titles of all references to find any relevant-sounding papers not already identified. For relevant titles, they also read through the abstracts and full papers. Snowballing resulted in 21 new items. Three of these papers were earlier versions of one of the publications we had initially identified. However, they had not appeared in our search results. The final set, including snowballing, included 49 papers. The first paper appeared in 2008 and the last paper was published in 2018.

c) Synthesise Methods: We identified a set of assessment criteria based on our own experience as well as criteria used in previous literature to evaluate papers [21]–[27]. One researcher then went through all the papers and extracted the methodology information. When uncertain, they discussed the outcome with the other researcher.

d) Synthesise Themes: Both authors reviewed the papers' main contributions in the final set and constructed an affinity diagram [28] to highlight the main contribution themes. Affinity diagrams are a grounded approach for sorting qualitative data into themes. In this case, we used the papers themselves as the unit of analysis and sorted them based on their primary topic. Both authors then completed a more in-depth reading of papers theme-by-theme resulting in several iterations to the themes and construction of sub-themes. While each paper is grouped under the theme associated with its primary contribution in Table II, many papers touch on multiple themes.

e) Limitations: We limited our initial query to 18 venues which we selected from Google Scholar. While we believe that this approach created a strong starting point, one could argue that a different selection approach might be more relevant and results in a more appropriate sample. To catch studies that we could not find in our initial search, we carried out a snowballing method to include more papers. We chose to limit our review to papers with user studies in them to both scale the review to a reasonable size and to focus on papers which take a deep look at the human factors issues developers face.

III. METHODOLOGY RESULTS

We begin by discussing the methodology information presented in the papers in terms of three main criteria: research design, data collection, and data analysis. This document comes with following supplementary materials: A BibTeX file, an Excel spreadsheet, and database queries. The files are available on the workshop's website and https://doi.org/10.7488/ds/2535.

TABLE II

List of all reviewed publications grouped by theme.

Publication/Theme	Study Method	Participants	N	Research Question	Pilot Case	Context	Recruitment	Demographics	Ethics	Mixed Methods	Data Analysis	Quotes	Compare with Literature	Limitations	Study Materials
Organisations and Context															
[29]	Semi structured interviews	Developers	15	○	○	○	◒	◒	○	○	◒	●	○	○	○
[30]	Semi structured interviews	Software professionals	42	○	○	○	●	●	◒	○	◒	●	●	●	○
[31]	Interviews	Developers	42	○	○	○	●	◒	◒	○	◒	○	◒	●	○
[32]	Survey, Survey	Developers	14, 61	○	●	○	●	◒	○	○	●	○	●	●	●
[33]	Survey, Survey, Observations, Interviews	Mix	15, 12, 23, 15	○	○	○	NA	○	○	●	◒	◒	○	○	○
[34]	Interviews	App security experts	12	●	○	○	●	◒	○	○	○	●	◒	○	○
[35]	Survey, Survey, Observations, Interviews	Mix	15, 12, 23, 15	●	○	◒	NA	●	○	●	●	●	◒	○	◒
[36]	Semi structured interviews	Mix (With crypto background)	21	●	○	◒	●	●	●	○	●	●	●	●	●
[37]	Semi structured interviews	Developers	13	●	○	○	●	●	●	○	●	●	●	●	●
[38]	Semi structured interviews	Security experts	32	◒	○	○	●	●	●	○	●	●	●	●	●
Structuring Software Development															
[39]	Semi structured interviews	Mix (70% Developers)	10	○	○	○	●	●	○	○	◒	●	●	○	○
[40]	Lab experiment (Architectural design task)	Students	90	●	○	○	◒	◒	○	○	●	○	◒	●	○
[41]	Online tasks	Developers	30	●	○	○	●	◒	○	○	●	○	◒	●	○
[42]	Lab experiment (Architectural design task)	Students	64	●	○	○	◒	◒	◒	○	●	○	◒	●	●
[11]	Survey, Lab experiment	Developers	295, 54	●	○	◒	●	●	●	●	●	○	◒	●	○
[43]	Survey	Software practitioners	9	●	○	○	○	○	○	●	○	○	◒	●	◒
[44]	Semi structured interviews, Survey, Survey	Developers	16, 51, 532	●	●	○	●	●	○	●	●	●	●	●	●
Privacy and Data															
[45]	Semi structured interviews, Online survey	Developers, Mix (58% Developers)	13, 228	○	○	◒	●	●	○	●	◒	●	●	○	○
[46]	Survey	Developers, Users	408 (267, 141)	●	●	●	●	◒	○	○	◒	●	◒	●	●
Third Party Updates															
[47]	Survey	Developers	203	●	○	○	●	●	●	●	◒	●	●	●	●
Security Tools Adoption															
[48]	Survey, Interviews, Lab study	FindBug users, FindBug users, Students	400, 12, 12	◒	○	◒	●	◒	○	●	○	○	◒	●	●
[49]	Survey	Developers	252	○	○	○	○	○	○	○	○	◒	○	○	○
[50]	Lab experiment	Students	9	●	○	◒	●	○	○	○	○	◒	◒	○	○
[51]	Lab experiment	Students, Developers	18, 9	●	○	◒	●	◒	○	○	◒	●	◒	●	○
[52]	Lab study	Developers	20	●	●	○	●	●	○	○	●	●	◒	●	○
[53]	Lab study (Programming)	Students	20	●	○	◒	◒	○	○	●	◒	●	◒	●	○
[54]	Think aloud	Students	8	●	○	○	○	○	○	○	○	●	◒	○	○
[55]	Lab experiment	Students	28	○	○	○	○	○	○	○	◒	●	◒	○	○
[56]	Lab study	Developers, Students	10 (5, 5)	●	●	○	◒	●	●	○	●	●	●	●	○
[57]	Field studies	Students	72	●	○	◒	◒	◒	◒	●	◒	●	◒	●	○
[58]	Interviews, Survey	Developers	5, 375	●	●	○	●	●	○	●	●	○	●	●	○
[59]	Cognitive walkthrough (CW), CW	Security experts, Developers	4, 4	○	○	○	○	○	○	○	○	●	◒	○	○
[60]	Observations	Developers	13	○	○	○	◒	◒	○	○	◒	●	◒	○	○
[61]	Task based	Developers (Academics, Professionals)	18 (9, 9)	◒	●	○	○	○	○	○	○	●	◒	○	●
[62]	Online programming task	Developers, Students	40 (16, 24)	○	●	○	●	●	●	○	●	○	◒	●	◒
[63]	Lab experiment (Programming)	Students	23	●	○	◒	●	○	○	○	○	●	◒	●	○
[64]	Online between subject	Developers	53	●	●	○	●	●	◒	○	◒	◒	●	●	●
Application Programming Interfaces															
[65]	Interviews	Developers	14	○	○	○	○	○	○	○	○	●	◒	●	○
[66]	Lab experiment, Programming tasks	Students	25	◒	○	○	○	◒	○	○	○	●	◒	●	○
[67]	Survey	Developers	47	◒	●	○	○	●	●	○	◒	●	●	○	○
[68]	Survey	Developers	45	○	○	○	○	○	○	○	○	●	◒	●	○
[69]	Survey, Survey	Developers	11, 37	●	○	◒	◒	●	○	●	◒	○	◒	●	●
[70]	Survey	Developers	55	○	○	◒	●	●	○	○	○	●	○	○	○
[71]	Online between subject	Developers	256	○	○	○	●	●	●	○	◒	○	◒	●	◒
[72]	Lab experiment (Programming)	Students	20	●	●	○	●	●	●	●	◒	●	◒	●	●
[73]	Online programming task	Developers (Professionals, Students)	109 (70, 39)	◒	○	◒	●	●	●	○	●	○	◒	●	○
Programming Languages															
[74]	Lab experiment	Developers	27	●	○	◒	●	◒	○	○	◒	○	◒	●	◒
Testing Assumptions															
[75]	Online between subject	Developers	307	○	○	○	●	●	●	○	◒	○	◒	●	●
[76]	Lab study (Programming)	Students	40	●	●	○	●	●	●	○	●	●	●	●	●

A full circle means that the paper has an explicit and clear statement for that criteria, a half circle means that there are some efforts to cover the metric (implied or partially covered), and an empty circle means the paper does not contain information on the specified criteria.

A. Research Design

Research Questions: Research questions show the purpose and outcome of the research [20] and are a vital component of an empirical research [21]. 'Why' and 'how' questions are preferred research questions in case studies [22], [77], such as "How do users respond to and perceive the code generation and explanation approaches?" [51, p. 3] and "Why do developers use CI [Continuous Integration]?" [44, p. 1]. Of our reviewed papers 26 explicitly state their research questions.

Pilots: When dealing with human subjects, it is prudent to conduct pilot studies before the main experiment [21], [24]. In software engineering studies that include human participants, it is advised to do pilot testing to ensure that software functions as expected during the experiment and that the tasks are clear [26]. Of our reviewed papers, 12 explicitly stated that they conducted a pilot study.

Context of the Study: Participants are effected by study context such as the lab setup, outside events, or their own expectations. Events occurring at the time of research can also impact participants' behaviour [23]. Of our reviewed papers 15 discuss the context of their study such as the time period and contextual information of the study. Only Sheth et al. gave a precise time period and elaborated on the events related to the study that might have influenced the results, i.e. NSA PRISM scandal in 2013 which could have influenced privacy concerns of participants during the study [46].

B. Data Collection

Sample and Population: All reviewed papers reported their sample size. On average, the number of participants in quantitative studies (surveys) was 195 (range: $9 - 532$, median: 55, std: 165.4), and in qualitative studies (interviews, lab experiments, observations, and online programming tasks), it was 38.8 (range: $4 - 307$, median: 20, std: 58.4).

Providing a detailed demographics of the participants assists the reader in understanding the context and also gives a better sense of the results. Of our reviewed papers, 12 provide no demographic information about the participants. The remaining 37 provide a mix of different demographic information ranging from basic age/gender, to more complex measurements of experience. In software engineering experiments it is recommend to report experience with programming languages, technologies related to the tool, industry, and natural language [26]. For instance, Acar et al. presented a detailed table for participants' demographics which gave information about invited/valid participants and some extra information from developers' Github profiles such as public repositories/gists and followers/followings [75].

Recruitment: Participant recruiting and sampling can impact generalisability and therefore the types of conclusions that can be drawn from the research. 29 papers reported their recruitment strategy clearly. For instance, Acar et al. state how they recruited and compensated developers. They ask Github developers to donate their time for research which proved to be a suitable method of recruiting developers [75].

Ethics: When dealing with human participants in research, it is necessary to treat them ethically. One method of doing so is to have an ethics review board that reviews and approves research plans in advance of research [78]. In our paper set, we notice that several studies do not explicitly report ethics and only 13 of papers explicitly mention that they have an ethics approval, e.g. institutional review board approval. Additionally, two papers mention collecting informed consent from participants but do not mention ethics approval, we recognised this as partial fulfilment of ethics criteria.

Mixed Methods: Gathering data from multiple sources, e.g. interviews and surveys, provides a richer view of the topic being studied [21]. 13 studies use a mixed method approach. For instance, Nadi et al.'s results take advantage of two artefact analyses and two surveys [69].

C. Data Analysis

Data Analysis Process: A detailed report of all steps researchers take increases the validity, reproducibilty, and gives a chance for other researchers to learn. For example, when the research involves qualitative analysis, it is often recommended to have at least two researchers work to analyse the data because one researcher could bring bias to the results [79]. However, in our sample, in several studies, data analysis is not described thoroughly, and in a few cases, the authors do not mention it explicitly at all. For example, Hilton et al. gave a link to the online codebook [44]. Additionally, 15 studies publicly shared all their study material. For instance, [44], [46], [69] provided a link to all their artefacts.

Quotes: Quotes bring evidence to the report [21]. Therefore if a study contains video, audio or written material from participants, it is beneficial to add quotes to the report. In our sample, 34 papers include quotes from participants. For example, Jain and Lindqvist incorporate relevant quotes in the text which fosters the results validity and also work as examples of how authors interpret the interviews [66].

Situate: Proper research highlights its position among other works, highlighting similarities and differences [21]. 31 of the reviewed papers discuss similar works in background; however, comparison of the results with other papers is sometimes overlooked. 14 papers compare their work with previous literature both in the background and discussion section.

Limitations: Study design decisions typically introduce limitations which need to be reported carefully so that other researchers can accurately draw conclusions from the research [27]. Bringing such issues to reader's attention increases the validity of the study as it gives more context and shows that the authors are aware of potential threats to their results [25]. Some common limitations in our sample are recruitment, small sample size, and generalisation of the results. 34 papers have an explicit section to elaborate on their limitations.

IV. Research Theme Results

A. Organisations and Context

Developers work within the larger context of organisations, teams, communities, and cultures. These social structures impact many elements of development from the types of tools a developer may be allowed to use by their organisation [30]–[32] to the assumptions of how to best handle non-functional requirements (NFRs) [35].

NFRs: Security is often referred to as a NFR in that it is expected to be included as part of high quality code development, but is rarely listed as an explicit requirement [35]. As a result, developers prioritise security below more-visible functional requirements or even easy-to-measure activities such as closing bug tracking tickets [30], [35], [37]. Pressure-related issues like budget and deadlines can also cause security to be prioritised lower [29]. To quote a participant from Poller et al. "If security is not on the list [of features], then is it really worth the time and extra energy to do it?" [35, p. 11]. The non-functional nature also made security challenging to assign as a task or decide who's job it is. This issue extended to tools such as libraries where developers assumed that issues like security were already correctly handled [37]. Managers generally view security as an important NFR, but expressing security downward is challenging to do without compromising team autonomy or creating more bureaucracy. A manager described it as one of the code quality "ilities" along with other NFRs such as usability, scalability, and maintainability [35].

Some organisations attempt to use external pressures such as penetration testing to motivate developers and help them understand the value of security, but without internal sustained support the motivation tends to lose priority compared to the functional requirement deadlines imposed on teams [30], [33], [35]. Similar to developers, managers are asked to make risky decisions and may choose to release code with known problems if doing so is aligned with business needs [38].

Clients do not necessarily prioritise security when providing feature lists unless the software they want has an obvious security focus, such as financial software, or the contracting organisation initiates discussions on the topic. This lack of guidance from clients forced developers to derive security guidance from the other functional requirements, or initiate a discussion about security needs with clients themselves [39].

Dedicated Security Teams: One issue is where to locate security within an organisation. Obviously it would be best if developers built security in from the start, but doing so requires a large amount of knowledge which takes time to learn and it can be difficult to self-motivate, especially when security is not seen as a measured functional requirement [33], [35]. Developers are also much more likely to learn about security because they are enthusiastic about it and want to know more, than if they are task driven [34].

Alternatively, security knowledge can be concentrated in a testing team or a set of security experts which act as a kind of roving source of security knowledge. While good at their jobs, these teams must convince others of the importance of security to get their changes made, they are also limited in their throughput and unable to minutely examine all generated code [38]. External penetration testing organisations can also be contracted to find security vulnerabilities. While effective at their primary task of finding problems, these penetration tests do not necessarily motivate developers to make long-term changes to their practices [33], [35]. Developers perceive security as the security team's problem and do not attempt to gain knowledge or install support tools on their own [30], [37].

Communication Around Fixing: Communication between testers, auditors, and developers is also a challenge. Security auditors consider a large part of their role to be motivating and convincing developers of the importance of identified vulnerabilities, especially when the vulnerability produced no visible problem [38]. Developers have difficulty seeing how a vulnerability could be practically exploited and find specific examples of actual attacks motivating [30]. Correcting vulnerabilities also required understanding it, which takes communication effort and occasionally results in dedicated training sessions run by the security team for often-observed problems [35], [38]. Security experts also struggled with having to communicate with many different teams, all of which have their own priorities and communication cultures [38].

Champions: Security champions are an often unofficial role held by someone on the development team who has limited security knowledge but considers security to be important and is willing to champion it. Champions are useful in getting security into products in a variety of ways. By putting champions in teams, managers can positively impact the security of a product without having to provide top-down pressure [35]. Security testers valued champions highly enough to find budget to send them to places like security conferences despite them being located on different teams [38].

Security Oriented Organisations: Structure and practices become different when companies focus heavily on security. These companies have a culture of security and they apply it in every step of development. They do not ignore security, or sacrifice it for the sake of releasing the product earlier. The customers in this market also demand security, and security is part of the culture and mindset of every entity involved in the process. Third party libraries are a point of discussion, some companies have to trust them because they are either certified by standard organisations or have been used by the community for a while, therefore they can trust them [36].

B. Structuring Software Development

Security Design Patterns: Security design patterns provide solutions for common problems faced during code design, such as how to best provide feedback to a user about password strength. Such patterns are widely used in HCI design, but currently are not as successful in security. Yskout et al. conducted a lab study with 32 teams of students, some with and some without access to the security pattern catalogue. Surprisingly, participants ended up with similar results in terms of productivity and security [42]. When the security pattern catalogue was annotated (security objective, applicability, trade-off labels, and relationships) participants found it easier to locate a suitable pattern [40].

Software Development Methodologies: There are many methodologies for how to "best" develop and deploy code as a holistic process, each of which have positive and negative impacts on security. Software development methodologies (SDMs) provide guidance on how to structure code development. Some SDMs, such as Agile, are feature focused where developers prioritise features each week and then work to get them implemented. The short iterations make it challenging to do full-stack testing on every iteration [39].

Continuous integration (CI) systems "automate the compilation, building, and testing of software" [44, p. 1] such that developers are encouraged to integrate their work frequently allowing for fast testing. CI is good in that it enables frequent automatic testing of code. However, it also faces practical problems such as needing to give the automated systems access to protected machines, the potential that the tested code is malicious, and the difficulty of allowing developers to run the same tests on their own computers which have less access. Hence, CI can introduce complexity and further security challenges to a project [44].

The fast nature of testing can also lead to prioritisation of automated tests over more manual tests such as penetration testing [43]. Manual code reviews can also be expensive and impractical as it needs several reviewers to look at a piece of code to find vulnerabilities [41]. While effective, automated tests can easily overlook unexpected issues, leading security experts to use both methods when reviewing code [38].

Information Sources: Information sources, such as documentation, are important to software developers especially when interacting with topics like APIs [69], [71]. To determine the effects of various information sources on code security, Acar et al. conducted a lab study comparing four information sources: official documentation only, StackOverflow only, book only, and free choice [11]. They found that participants that used StackOverflow were more likely to create functional code, but less likely to produce secure code.

C. Privacy and Data

Privacy is a complex topic and how people generally manage it is a well researched topic outside the scope of our research. However, privacy as a software requirement has only been marginally studied. Sheth et al. conducted a survey of Europeans and North Americans and found that Europeans were significantly less willing to give up privacy for functionality [46]. They also find that "data privacy is often an implicit requirement: everyone talks about it, but no one specifies what it means and how it should be implemented [...] While almost all respondents agree about the importance of privacy, the understanding of the privacy issues and the measures to reduce privacy concerns are divergent" [46, p. 9].

Developers are also not always aware of software privacy issues, nor do they endanger users' privacy deliberately. Balebako et al. showed that only $1/3$ of App developers ($n = 228$) knew what data was being collected by third-party tools [45]. Revenue models are a determinant of how much data developers collect which means if an app's revenue model is ad-based, it is likely to collect more data than a paid app. Company size is also a player in privacy; smaller companies tend to be less privacy-conscious [30], [45].

D. Third Party Updates

When vulnerabilities are discovered in software, developers (hopefully) fix them and release an update to their code. Like any other software, libraries can have vulnerabilities and updates to address those vulnerabilities. One potential source of insecure software happens when a library author updates their code, but developers who use that library do not switch to the updated library. An analysis of over a million apps showed that 85.6% of the libraries could be updated by simply changing the version number of the library [47]. Developers avoid updating libraries primarily because the update may break their app or require them to spend time adjusting to new library structures. When they want to understand potential library changes, they use change logs [47].

E. Security Tool Adoption

Software is commonly written with the assistance of tools such as Integrated Development Environments (IDEs). Several research papers have endeavoured to understand the tool needs of developers, improve existing tools to better support security, or build new security-focused tools.

Security tools generally see poor adoption by developers. To understand why, researchers have surveyed [32], [48], [49], [58] and interviewed [30], [48], [52], [58], [60] developers and auditors. They find that organisation and team policies are a driving factor to tool adoption [58], though many organisations do not encourage their use [48], [49], larger organisations make more use of security tools than small ones [30], and peer tool use positively impacts use [30], [32]. Though, surprisingly, having more concern about security did not lead to greater security tool usage, but having an academic background or training in security did [32]. Existing tools also exhibit pain points by checking for the wrong types of problems by default, having poor warning messages [48], [58], interrupting work flow [52], [58], [60], having too many false positives [52], [58], not providing enough support for team work, and integrating poorly with IDEs [52]. When using tools, developers want to know about the type of attacks, available solutions, vulnerabilities, and flow of data in their programs [56]. Visualising tool output also helps [59].

Several works focus on the creation and evaluation of specific tools which are often built as plugins for popular IDEs. FindBugs is a static analysis tool that looks for coding defects by analysing software in the abstract, allowing it to identify issues at the logical level. It has been the subject of multiple research projects which considered both the context of use and usability needs in its design and evaluation [48], [49], [56]. New tools which focus on usability, also use it as a staring point and a basis for comparison [59].

The Application Security IDE (ASIDE) is a static analysis Eclipse plugin helps developers find potentially vulnerable web application code "in situ" as they code, similar to the underline in a word processing spell checker [50]. The initial user study with students had mixed results [50], but a larger follow-on study found students using ASIDE improved in their post-usage code security knowledge [53]. Another ASIDE study compared giving graduate students auto generated code fixes to giving them explanations of identified vulnerabilities [51]. They found that students given code solutions were more likely to implement them. ASIDE was then further improved to allow developers to annotate security-critical code so that the tool could provide better analysis [54], [55]. However, they found that student participants did not have enough security knowledge to accurately provide the annotations.

Two other studies also proposed and tested fast-feedback tools. CHEETAH provides feedback-on-compile warnings to developers using an IDE static taint analysis plugin [61]. Their study compared their feedback-on-compile approach to feedback-on-request and found that developers both preferred the feedback-on-compile approach and that they fix bugs faster. However, the feedback-on-request also introduced a time delay, possibly confounding results. FixDroid is an IDE plugin for Android developers that highlights insecure code, and on mouse over provides a short explanation and recommendation to fix the problem [62]. They find that developers using FixDroid's feedback produced more secure code.

Gorski et al. designed an online programming experiment (study design modelled on [71]) with Python developers to find out whether showing developers API-integrated security warnings would help them with security APIs [64]. These security warnings include a warning and an example of a secure code snippet. Developers who had access to warnings and code examples created more secure code compared to who did not have access to warnings.

Education: We did not explicitly search for educational approaches in security; however, three paper showed up in our results. Tabassum et al. investigated two approaches to teaching security and secure coding, teaching assistant vs. tools [63]. Educational Security in the IDE (ESIDE) is an Eclipse IDE plugin, which provide fast feedback warnings, detailed explanations, and suggested remediation code. The user study had some serious limitations, but generally found that students found ESIDE interesting, used the suggested code samples, but did not engage with the educational information and did not try and fix errors on their own [63]. Two other papers look at the potential benefits of a similar tool in teaching secure coding [53], [57].

F. Application Programming Interfaces (APIs)

Considering Options: While designing and coding, developers must make decisions about many features, including security and privacy. Many external pressures, like deadlines often overload developers, quality and functionality and security is often not a part of developers' decision-making process. Security is often overlooked because it needs extra effort and it is 'blind spot' in developers mindset [67]. API blind spots are the points where developers incorrectly use an API, often because they just trust the API to do the right thing without doing additional checks [73]. However, if developers are nudged with the security, they change their programming approach and consider security and secure programming [67], [72]. The involvement of customers in security also impacts decision making, and while customers are often unclear about their security requirements, developers felt that customer engagement on security topics was helpful [29], [39].

The features provided by readily available APIs can also impact security and privacy choices of developers. In a case of geo-location libraries, developers who have access to a library with privacy-preserving options are more willing to use coarse location information over those with only access to the standard library [66].

Testing the Usability of Security APIs: Security APIs (SecAPIs), as a subset of APIs, are built to help developers with security concepts such as encryption and authentication. They offer a level of abstraction and work as a layer between developers and the low-level details that developers may get wrong on their own. SecAPIs broadly fall into two categories, *security primitives* which required security knowledge to understand and *security controls* which were found to be a more appropriate abstraction level for developers. Unlike other APIs, SecAPIs do not well support learning-by-doing [70].

Acar et al. compared the usability of five Python cryptographic APIs and found that good documentation with examples is more helpful to developers than just having a simple API [71]. They also found that when developers do not find a solution easily, they tend to look online, and typically find an insecure solution on StackOverflow. Resulting in the concerning situation where developers think they have a secure solution, but actually have an insecure one. Another study similarly found that developers struggle to use SecAPIs finding them overly complex for basic tasks and they want better documentation and higher abstraction levels [69].

Transport Layer Security (TLS) and its predecessor Secure Sockets Layer (SSL) are common protocols to encrypt data in transit. Unfortunately, developers have difficulty using these protocols correctly [65]. Fahl et al. tried notifying app developers of TLS-related vulnerabilities in their apps [65]. However, after three months 34.6% had not made corrections. Similarly, Oltrogge et al. found that developers found TLS pinning too complex of a topic to use [68].

G. Programming Languages

Only one paper conducted a laboratory study of the relative usability of different programming languages (PLs). Undoubtedly more such studies exist based in artefact analysis, but Prechelt's study is notable in that: 1) all the development teams had the same brief, 2) PLs were assigned (Java, PHP, Pearl) and 3) developers had two days to complete the brief [74]. Theoretically, this design allows for a more causation-style analysis than artefacts provide. However, they find no definitive links between PLs and security, suggesting that security issues may be more develop-based than language-based.

H. Testing Assumptions

While several developer studies use students as their subjects, it is important to know if this group is representative of professional developers. In a Github sample, status (being a student or a professional) and security background were not a significant factor in terms of the security and functionality of the final code, and the only distinguishing feature was years of experience [75]. In contrast, a different study observed no impact of years of experience on security [41], [76]. These studies provide an initial suggestion that students might be representative of developers with a similar level of experience.

Naiakshina et al. ran a lab study (similar to their 2017 study [72]) with forty CS students to find out if priming effected the production of secure code [76]. In their 2017 study they used an interview to gather data from participants, and in their 2018 study they used a survey instead of interviews. Results show that interviews generated valuable results, particularly in developer studies because this field is still at its early stages, so interviews allow for more flexibility in data collection. An interesting side takeaway is that PL experience is not a deciding factor in developers security which contradicts with findings in [75]. Another contradictory result is around copy/pasting behaviour. Acar et al. show that copy/pasting results in insecure code [11], but [76] observed that using copy/paste tended to result in more secure code.

V. Discussion

A. Methodology and Ability to Generalise

Developers are a challenging group to study because their work is undertaken over an extended time frame, collaborative, involves multiple stake holders, and requires decision making based on a combination of prior experience and online research. These features make it challenging to create a lab or online study that properly replicates the experience of software development without making it long and expensive. To handle these problems we see a heavy reliance on retrospective and opinion studies like surveys and interviews where developers can reflect on past work. For lab studies involving tools, we see efforts to use code that the developer is already familiar with possibly as a way to better evaluate the tool [53], [62]. However, in our opinion, some of these efforts worked poorly, such as asking professional developers to write a class example as a way of getting them to write something small, but as a side effect they also explicitly left out some checks because adding them would be confusing to students [51].

Length of the study was also a tricky subject. Longer lab studies took as much as 3 days [74] and shorter ones took as little as $15-20$ minutes [63]. Most of the ASIDE studies lasted for about 3 hours and had developers use their coursework as the code base to make starting easier. Our impression from reading many such papers, is that studies that allowed less than 3 hours tended to suffer from the developers not having enough time to really get involved with the programming or interact with the tool in a realistic way.

As a field, DCS is relatively young and many of the methodologies have not caught up to the standards of HCI research. For example, several methodologies mentioned conducting a think aloud study where they interrupted the participant whenever they did something interesting. While that is an acceptable approach in industrial research where the goal is to find big problems, in academic HCI research interrupting the participant mid-task is known to impact their behaviour and disrupt their work flow which invalidates later results. It was also surprising how few studies attempted to compare their results to a control or similar tool on the same tasks. Acar et al. did this by making the control be a book [11]. But many other studies only tested their new tool and made no attempt to even compare against other tools on the same task. Much less base their tasks to other study setups.

B. Research Gaps

We identified several research gaps while reviewing that we feel are important to fill, though this list is hardly exhaustive.

a) When to Interrupt the User: The majority of the tools in our reviewed papers took the view that providing fast feedback *in-situ* was a good thing because it would give developers a chance to make changes right away. However, this assumption is only weakly tested in [61] which looked at feedback-on-compile vs. feedback-on-request. But given their confounds, the question is still open. The assumption is also at odds with how humans write text and when it is best to interrupt them doing so [80]. Best practice in that area suggests waiting till a user has reached the end of a text passage before interrupting with feedback. But the equivalent of finishing a text passage in code is not necessarily clear at this stage.

b) Are Students Similar to Professional Developers?: Many papers use students as participants under the assumption that they are similar to real software developers (and easier to find). However, there were only two studies [75], [76] in our set that compared students to developers and their findings disagree on several points. Shortly after we completed our review, Naiakshina et al. released a new paper [81] which roughly duplicated their earlier work on password storage conducted with student participants [72]. They find that students and developers have difficulty securing passwords and that paying more for developers does not improve the situation. While this lends some support to the view that students are similar to professional developers, the issue is still open.

c) Tools: While there is abundance of work in security tool adoption (Section IV-E), the work focuses on only a few IDE plugins, yet several other security plugins are available [82] which could benifit from usability analysis. For example, SpotBugs is a successor of FindBugs, and comparing it with FindBugs [83] could be quite interesting, especially given that most papers focus on only a single tool. In relation to Section IV-A, there is also a question of how these tools, e.g. static analysis tools, integrate with developers' workflows, and how these tools can be used in organisations.

d) Testing Support for Team Development: No study in our review tried simulating team or collaborative aspects of programming. Several studies commented on the frustrations developers have with tools that do not well support team work. While this concern is minimally elaborated on, issues like having to share the particulars of a code problem with the security team or needing to add output to a bug tracker come up elsewhere. It would be interesting to see studies of how developers communicate security issues in teams and the types of details that need to be shared and tracked.

e) Learning Support: Due to online resources having a negative impact on code security [11], offering reliable documentation to developers in real-time as an IDE plugin could be beneficial. Tools such as ESIDE [63] and FixDroid [62] do provide educational feedback which tends to be ignored in studies by task-driven developers, and example solutions which developers do appear to use. However, both of these works are initial case studies and do not look at different warning texts, styles or even the appropriate length of education to provide to developers. Links to "good" external resources are also not tried.

f) Privacy Support: Privacy features can be complicated to define and include in software requirements. Efforts to bring privacy into design [84], [85] are currently under research though more research is required as it is a hot-button issue, especially with the recent passing of GDPR in the EU [86]. Privacy perceptions and security mindsets effect decisions (see [87] for average users' mental models in security). This correlation becomes more relevant when developers are responsible for building tools that can impact people's day to day lives. More research is needed to find out privacy mindsets of software developers and how to support them in developing privacy aware tools for the general public [88], [89]. Cultural differences are another major theme here. International companies treat the data and privacy of users around the globe based on their home country. However, a single geographical area is not a representative sample of all countries [90]. Hence, more research is needed to learn about how software developers and companies can adapt their technologies to various cultures.

VI. Conclusions

This paper reports a systematic literature review of 49 research papers that are DCS related. Every study has at least one user study of software developers. We discuss our sample set from two viewpoints, methodology and findings. In the former, we look at research design, data collection, and data analysis of papers as a sample set. We observed similar issues in DCS research addressed in the literature [25], [26]. In the latter, we synthesis outcomes of all studies. Eight themes emerge from our dataset: organisations and context, structuring software development, privacy and data, third party updates, security tool adoption, application programming interfaces, programming languages, and testing assumptions. Our results facilitate entry of early researchers to the field of DCS and assist research veterans in discovering areas that are not yet researched thoroughly and need more investment.

Acknowledgement

We thank Kholoud Althobaiti, Tariq Elahi, Adam Jenkins, Maria Wolters, and everyone associated with the TULiPS-Lab at the University of Edinburgh for helpful discussions and feedback. We also thank the anonymous reviewers and our shepherd whose comments helped improve the paper greatly. This work was sponsored in part by Microsoft Research through its PhD Scholarship Programme.

References

[1] K. Munro. (2015) Hacking kettles & extracting plain text wpa psks. yes really! [Online]. Available: https://www.pentestpartners.com/security-blog/hacking-kettles-extracting-plain-text-wpa-psks-yes-really/

[2] G. Liang *et al.*, "The 2015 Ukraine Blackout: Implications for False Data Injection Attacks," *IEEE Transactions on Power Systems*, 2017.

[3] G. Martin *et al.*, "Cybersecurity and healthcare: how safe are we?" *BMJ*, 2017.

[4] A. Van der Stock *et al.*, "Top 10 Most Critical Web Application Security Risks," The OWASP Foundation, Tech. Rep., 2017.

[5] Veracode, "State of Software Security," Veracode, CA Technologies, Tech. Rep., 2018. [Online]. Available: https://www.veracode.com/state-of-software-security-report/

[6] M. Egele *et al.*, "An empirical study of cryptographic misuse in android applications," in *CCS*. ACM, 2013.

[7] A. Adams *et al.*, "Users Are Not The Enemy," in *Commun. ACM*, 1999.

[8] M. A. Sasse *et al.*, "Usable security: Why do we need it? How do we get it?" in *Security and Usability Designing Secure Systems that People Can Use*. O'Reilly, 2005.

[9] G. Wurster *et al.*, "The Developer is the Enemy," in *Proc. of the 2008 NSPW*. ACM, 2008.

[10] M. Green *et al.*, "Developers are Not the Enemy!: The Need for Usable Security APIs," *IEEE Security Privacy*, vol. 14, no. 5, Sep. 2016.

[11] Y. Acar *et al.*, "You Get Where You're Looking for: The Impact of Information Sources on Code Security," in *IEEE SP*, 2016.

[12] O. Pieczul *et al.*, "Developer-centered Security and the Symmetry of Ignorance," in *Proc. of the 2017 NSPW*. ACM, 2017.

[13] B. A. Myers *et al.*, "Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools," *Computer*, 2016.

[14] A. J. Ko *et al.*, "The State of the Art in End-user Software Engineering," *ACM Comput. Surv.*, vol. 43, no. 3, Apr. 2011.

[15] L. S. Amour *et al.*, "Improving Application Security through TLS-Library Redesign," in *Security, Privacy, and Applied Cryptography Engineering*. Springer, 2015.

[16] O. Tripp *et al.*, "ALETHEIA: Improving the Usability of Static Security Analysis," in *CCS*. ACM, 2014.

[17] G. Pellegrino *et al.*, "jÄk: Using Dynamic Analysis to Crawl and Test Modern Web Applications," in *Research in Attacks, Intrusions, and Defenses*. Springer, 2015.

[18] J. Wäyrynen *et al.*, "Security Engineering and eXtreme Programming: An Impossible Marriage?" in *Extreme Programming and Agile Methods - XP/Agile Universe*. Springer, 2004.

[19] K. Vaniea *et al.*, "Tales of software updates: The process of updating software," in *CHI*, 2016.

[20] C. Okoli, "A Guide to Conducting a Standalone Systematic Literature Review," *CAIS*, 2015.

[21] L. Dub *et al.*, "Rigor in Information Systems Positivist Case Research: Current Practices, Trends, and Recommendations," *MIS Quarterly*, 2003.

[22] P. Runeson *et al.*, *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.

[23] K. Krol *et al.*, "Towards robust experimental design for user studies in security and privacy," in *LASER*, 2016.

[24] R. Maxion, *Dependable and Historic Computing*. Springer, 2011, ch. Making Experiments Dependable.

[25] K. P. L. Coopamootoo *et al.*, *Cyber Security and Privacy Experiments: A Design and Reporting Toolkit*. Springer, 2018.

[26] A. J. Ko *et al.*, "A practical guide to controlled experiments of software engineering tools with human participants," *Empirical Software Engineering*, 2015.

[27] S. Schechter, "Common Pitfalls in Writing about Security and Privacy Human Subjects Experiments, and How to Avoid Them," Microsoft, Tech. Rep., January 2013.

[28] C. Courage *et al.*, "Appendix F - Affinity Diagram," in *Understanding Your Users*, ser. Interactive Technologies. Morgan Kaufmann, 2005.

[29] Jing Xie *et al.*, "Why do programmers make security errors?" in *VL/HCC*, 2011.

[30] S. Xiao *et al.*, "Social Influences on Secure Development Tool Adoption: Why Security Tools Spread," in *CSCW*, 2014.

[31] J. Witschey *et al.*, "Technical and Personal Factors Influencing Developers' Adoption of Security Tools," in *ACM Workshop on SIW*, 2014.

[32] ——, "Quantifying Developers' Adoption of Security Tools," in *ESEC/FSE*, 2015.

[33] S. Türpe *et al.*, "Penetration Tests a Turning Point in Security Practices? Organizational Challenges and Implications in a Software Development Team," in *SOUPS*, 2016.

[34] C. Weir *et al.*, "How to Improve the Security Skills of Mobile App Developers? Comparing and Contrasting Expert Views," in *SOUPS*, 2016.

[35] A. Poller *et al.*, "Can Security Become a Routine?: A Study of Organizational Change in an Agile Software Development Group," in *CSCW*, 2017.

[36] J. M. Haney *et al.*, ""We make it a big deal in the company": Security Mindsets in Organizations that Develop Cryptographic Products," in *SOUPS*, 2018.

[37] H. Assal *et al.*, "Security in the Software Development Lifecycle," in *SOUPS*, 2018.

[38] T. W. Thomas *et al.*, "Security During Application Development: An Application Security Expert Perspective," in *CHI*. ACM, 2018.

[39] S. Bartsch, "Practitioners' Perspectives on Security in Agile Development," in *ARES*, 2011.

[40] K. Yskout *et al.*, "Does Organizing Security Patterns Focus Architectural Choices?" in *ICSE*, 2012.

[41] A. Edmundson *et al.*, "An Empirical Study on the Effectiveness of Security Code Review," in *ESSoS*, 2013.

[42] K. Yskout *et al.*, "Do Security Patterns Really Help Designers?" in *ICSE*, 2015.

[43] A. A. Ur Rahman *et al.*, "Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices," in *CSED*, 2016.

[44] M. Hilton *et al.*, "Trade-offs in Continuous Integration: Assurance, Security, and Flexibility," in *ESEC/FSE*, 2017.

[45] R. Balebako *et al.*, "The Privacy and Security Behaviors of Smartphone App Developers," in *Proc. of Workshop on Usable Security*, 2014.

[46] S. Sheth *et al.*, "Us and Them: A Study of Privacy Requirements Across North America, Asia, and Europe," in *ICSE*, 2014.

[47] E. Derr *et al.*, "Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android," in *CCS*. ACM, 2017.

[48] N. Ayewah *et al.*, "A report on a survey and study of static analysis users," in *Workshop on Defects in large software systems*, 2008.

[49] ——, "Using Static Analysis to Find Bugs," *IEEE Software*, 2008.

[50] J. Xie *et al.*, "ASIDE: IDE support for web application security," *Proc. of the 27th Annual Computer Security Applications Conference*, 2011.

[51] ——, "Evaluating Interactive Support for Secure Programming," in *CHI*, 2012.

[52] B. Johnson *et al.*, "Why don't software developers use static analysis tools to find bugs?" in *ICSE*, 2013.

[53] J. Zhu *et al.*, "Interactive support for secure programming education," in *SIGCSE*. ACM Press, 2013.

[54] ——, "Supporting secure programming in web applications through interactive static analysis," *J. Adv. Res.*, 2014.

[55] T. Thomas *et al.*, "A study of interactive code annotation for access control vulnerabilities," in *IEEE VL/HCC*, 2015.

[56] J. Smith *et al.*, "Questions Developers Ask While Diagnosing Potential Security Vulnerabilities with Static Analysis," in *ESEC/FSE*, 2015.

[57] M. Whitney *et al.*, "Embedding Secure Coding Instruction into the IDE: A Field Study in an Advanced CS Course," in *SIGCSE*, 2015.

[58] M. Christakis *et al.*, "What developers want and need from program analysis: an empirical study," in *ASE*, 2016.

[59] H. Assal *et al.*, "Cesar: Visual representation of source code vulnerabilities," in *IEEE VizSec*, 2016.

[60] T. W. Thomas *et al.*, "What Questions Remain? An Examination of How Developers Understand an Interactive Static Analysis Tool," *SOUPS*, 2016.

[61] L. N. Q. Do *et al.*, "Just-in-time static analysis," in *Proc. of ISSTA*, 2017.

[62] D. C. Nguyen *et al.*, "A Stitch in Time: Supporting Android Developers in Writing Secure Code," in *CCS*, 2017.

[63] M. Tabassum *et al.*, "Comparing Educational Approaches to Secure Programming : Tool vs. TA," *SOUPS*, 2017.

[64] P. L. Gorski *et al.*, "Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse," in *SOUPS*, 2018.

[65] S. Fahl *et al.*, "Rethinking SSL Development in an Appified World," in *CCS*, 2013.

[66] S. Jain *et al.*, "Should I Protect You? Understanding Developers' Behavior to Privacy-Preserving APIs," in *Workshop on USEC*, 2014.

[67] D. Oliveira *et al.*, "It's the psychology stupid: how heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots," in *ACSAC*, 2014.

[68] M. Oltrogge *et al.*, "To Pin or Not to Pin Helping App Developers Bullet Proof Their TLS Connections," *USENIX Security*, 2015.

[69] S. Nadi *et al.*, "Jumping Through Hoops: Why Do Java Developers Struggle with Cryptography APIs?" in *ICSE*. ACM, 2016.

[70] L. Lo Iacono *et al.*, "I Do and I Understand. Not Yet True for Security APIs. So Sad," in *European Workshop on Usable Security*, 2017.

[71] Y. Acar *et al.*, "Comparing the Usability of Cryptographic APIs," in *IEEE Symposium on Security and Privacy*, 2017.

[72] A. Naiakshina *et al.*, "Why Do Developers Get Password Storage Wrong?: A Qualitative Usability Study," in *CCS*, 2017.

[73] D. S. Oliveira *et al.*, "API Blindspots: Why Experienced Developers Write Vulnerable Code," in *SOUPS*, 2018.

[74] L. Prechelt, "Plat_forms: A Web Development Platform Comparison by an Exploratory Experiment Searching for Emergent Platform Properties," *IEEE Transactions on Software Engineering*, 2011.

[75] Y. Acar *et al.*, "Security Developer Studies with GitHub Users: Exploring a Convenience Sample," *SOUPS*, 2017.

[76] A. Naiakshina *et al.*, "Deception Task Design in Developer Password Studies: Exploring a Student Sample," in *SOUPS*, 2018.

[77] R. K. Yin, *Case study research and applications: Design and methods*. Sage Publications, 2018.

[78] M. Saunders *et al.*, *Research Methods for Business Students*. Prentice Hall, 2012.

[79] J. Lazar *et al.*, "Interviews and focus groups," in *Research Methods in Human Computer Interaction (2nd Edition)*. Morgan Kaufmann, 2017.

[80] L. Flower *et al.*, "A cognitive process theory of writing," *College composition and communication*, 1981.

[81] A. Naiakshina *et al.*, ""if you want, i can store the encrypted password." a password-storage field study with freelance developers," in *CHI*, 2018.

[82] A. Z. Baset *et al.*, "IDE Plugins for Detecting Input-Validation Vulnerabilities," in *IEEE Security and Privacy Workshops*, 2017.

[83] J. Smith *et al.*, "How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool," *IEEE Transactions on Software Engineering*, 2018, Early Access version.

[84] M. Langheinrich, "Privacy by Design-Principles of Privacy-Aware Ubiquitous Systems," in *Ubiquitous Computing*, 2001.

[85] I. Hadar *et al.*, "Privacy by designers: software developers' privacy mindset," *Empirical Software Engineering*, 2018.

[86] EU GDPR Information Portal. [Online]. Available: https://www.eugdpr.org/

[87] R. Wash, "Folk models of home computer security," in *SOUPS*, 2010.

[88] A. Senarath *et al.*, "Why Developers Cannot Embed Privacy into Software Systems?: An Empirical Investigation," in *EASE*, 2018.

[89] A. R. Senarath *et al.*, "Understanding user privacy expectations: A software developer's perspective," *Telematics and Informatics*, 2018.

[90] Y. Sawaya *et al.*, "Self-Confidence Trumps Knowledge: A Cross-Cultural Study of Security Behavior," in *CHI*. ACM, 2017.