One Size Does Not Fit All: A Grounded Theory and Online Survey Study of Developer Preferences for Security Warning Types

Anastasia Danilova
University of Bonn
danilova@cs.uni-bonn.de

Alena Naiakshina
University of Bonn
naiakshi@cs.uni-bonn.de

Matthew Smith
University of Bonn, Fraunhofer FKIE
smith@cs.uni-bonn.de

ABSTRACT

A wide range of tools exist to assist developers in creating secure software. Many of these tools, such as static analysis engines or security checkers included in compilers, use warnings to communicate security issues to developers. The effectiveness of these tools relies on developers heeding these warnings, and there are many ways in which these warnings could be displayed. Johnson et al. [46] conducted qualitative research and found that warning presentation and integration are main issues. We built on Johnson et al.'s work and examined what developers want from security warnings, including what form they should take and how they should integrate into their workflow and work context. To this end, we conducted a Grounded Theory study with 14 professional software developers and 12 computer science students as well as a focus group with 7 academic researchers to gather qualitative insights. To back up the theory developed from the qualitative research, we ran a quantitative survey with 50 professional software developers. Our results show that there is significant heterogeneity amongst developers and that no one warning type is preferred over all others. The context in which the warnings are shown is also highly relevant, indicating that it is likely to be beneficial if IDEs and other development tools become more flexible in their warning interactions with developers. Based on our findings, we provide concrete recommendations for both future research as well as how IDEs and other security tools can improve their interaction with developers.

CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in HCI**;
• **Security and privacy** → *Usability in security and privacy.*

KEYWORDS

developer security warnings, software development, code security

ACM Reference Format:
Anastasia Danilova, Alena Naiakshina, and Matthew Smith. 2020. One Size Does Not Fit All: A Grounded Theory and Online Survey Study of Developer Preferences for Security Warning Types. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380387

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05.
https://doi.org/10.1145/3377811.3380387

1 INTRODUCTION

Security warnings are a form of a computer dialog communication, e.g., used to notify users about potential risks of their actions and security issues [8]. Research into human interaction with security warnings has been studied for decades in cases such as browser warnings and phishing alerts for end users. This research shows that end users often struggle with security warnings, but that taking human factors into account can improve adherence to these warnings [6, 7, 13–15, 17, 31, 33–35, 58, 64–66]. Software developers can also encounter security warnings while programming and, just like end users, can become frustrated by these messages. Examples of severe security issues in software development are the use of deprecated security parameters or functions for end user password storage in a database endangering a large amount of sensitive user data [4, 52–54] or the misuse of TLS enabling man-in-the middle attacks [32]. Consequently, improving security warnings for developers is a desirable research goal [43].

Many tools can be used by developers to spot potential security issues, such as compilers or static and dynamic code analyzers (e.g., [12, 47, 55]) and there is already work underway looking at these different types of systems. For instance, recent work by Gorski et al. has shown that warnings shown by the compiler can improve code security [42]; however, Barik et al. showed that reading compiler warnings takes a significant amount of effort and that improvements are needed [11, 12].

In the realm of static analysis, Johnson et al. conducted a qualitative study to examine why developers don't use static analysis tools. They found that the way in which warnings are presented, lack of fixing help and the poor integration of warnings into developers' workflows has a detrimental effect on the use of such supporting tools [46].

In this paper we build on Johnson et al.'s work and examine developers' wishes concerning how and when they would like to receive security warnings. There are many different ways warnings can be shown, e.g., in the compiler output, with underlines and warning markers in the code window, in a stand-alone security view, or as pop-ups. In the rest of the paper we will refer to these as different warning *types*. Warnings can also be shown at different times during the development process. To help inform how tools design warning message interaction, we conducted research on these two aspects of warnings for developers. We specifically did not study the content of the warnings, the underlying accuracy of the warnings (i.e., false positive rate), or additional features such as quick fixes, since these can be applied to all warning types and are orthogonal to our research.

To this end, we conducted a Grounded Theory (GT) study with 14 professional software developers and 12 computer science students, to build a working theory as to how developers would like to interact with warnings. We compared the following security warning types which could be shown in an IDE: markers in the code view, compiler output, dedicated security view, general warning view, pop-up warnings, and warnings on committing. As it is recommendable when doing this kind of qualitative work, we triangulated the early results by using a focus group with 7 academic researchers to include new themes into the GT study. To gather quantitative insights into the theories developed from our qualitative research as well as selected themes from Johnson et al.'s qualitative work, we ran an online survey with 50 professional software developers.

The rest of the paper is organized as follows: First, we discuss related work (Section 2) and our methodology in detail (Section 3). Then, we present our explanatory theory in Section 6, whereas our quantitative analysis can be found in Section 7. In Section 8 we discuss the implications of our study and provide recommendations for security warning design for developers.

2 RELATED WORK

Barik et al. [12] conducted an eye-tracking study with student developers to investigate Java compiler error messages. They examined different compiler errors in the Eclipse IDE and found that developers needed slightly longer to read and understand an error message than writing source code. They also found that developers actually read error messages. In a further study, Barik et al. [11] investigated compiler errors and how developers understood error messages. They conducted a comparative study of 68 software developers for two compilers (Jikes and OpenJDK) for Java and examined 210 compiler errors from Stack Overflow. In their study, Barik et al. focused on the warning message and argumentative structure and found that developers preferred errors with a clear argumentative structure of the error or with solutions to the problem.

Wogalter et al. [27] deployed the communication-human information processing (C-HIP) model describing the interaction between computers and humans for warnings and risk communication. This model specified that communication components, like attitudes and beliefs of the receiver, play a role in communication. Furthermore, Wogalter et al [67] summarized studies on general warning design and provided examples to increase the visibility of a warning, such as coloring, wording, and layout. They also mentioned the location of warnings as a crucial factor.

Gorski et al. [42] investigated the effect of python compiler warnings on preventing security API misuse. They found that developers improved their code after seeing the compiler security warnings. They stated that API designers could include warnings in their design to trigger compiler warnings.

Moreover, many tools and static analyzers were introduced to help developers manage security issues. For instance, Christakis and Bird [23] surveyed 375 developers from Microsoft to investigate their perceptions of automatic program analysis. Their results summarized the characteristics and functionality a program analyzer should have. They found that false positives should be limited to 15-20%. Furthermore, the majority of their participants wanted program analyzer output to appear in the editor instead of the build output or code review. Therefore, we concentrated on warnings for IDEs and editors.

Nguyen et al. [55] proposed the plugin FixDroid, which offered warnings and quick fix dialogues for security issues in the Android Studio IDE, and found that developers approved FixDroid, which helped produce more secure code.

Smith et al. [63] conducted an exploratory study with 10 novice and experienced software developers to explore how they resolve security defects while using static analysis tools. Participants used Find Security Bugs and requested more information on security vulnerabilities, associated attacks, and fixes, but also on the software and its social ecosystem, on related resources and tools. Smith et al. gave recommendations on how future tools could leverage their findings to facilitate better strategies, e.g., the integration of a context-informed web search with a specialized search engine.

Johnson et al. [46], who investigated the reasons why developers rarely used static analyzers, found that large volumes of warnings, false positives, and poor warning presentation caused developers to stop using analyzers. Furthermore, Johnson et al. [45] conducted a qualitative study, in which 26 participants interacted with warnings from FindBugs, the Eclipse Compiler, and EclEmma. The authors presented their resulting communication theory, which revealed understanding the notifications posed multiple challenges, which are caused by gaps and mismatches between developers' programming knowledge and communication methods used by the notifications. In addition to that, the authors concluded that expectation mismatch challenges focus on visual communication and that tools can improve how they communicate to developers if they would respect developers' knowledge and experiences.

The related work above examines many different aspects of security warnings for developers, e.g., how to improve compiler warnings, how developers interact with static code analyzers, how to improve static code analyzer warnings, etc. We add to this work by broadening the scope and examining which form of warnings developers might prefer, e.g., would they prefer to receive warnings via the compiler or from a static code analyzer while they are programming or via a pop-up warning before they commit code.

Furthermore, Bailey et al. [10] found that participants performed worse on interrupted than on non-interrupted tasks. They recommended to notify the user in an appropriate moment in order to limit the disruptive effect on the main task. Hudson et al. [44] presented promising results by using sensors (audio and video recording) to form statistical models to predict the interruptibility of users in order to assess when a task can be interrupted. Robertson et al. [59, 60] compared the effects of two interruption styles on end-user programmers: *negotiated-style* and *immediate-style* interruptions. They found that negotiated-style interruptions were more accepted than immediate reactions [60] but can differ in their intensity. In their follow-up study Robertson et al. found that high-intensity negotiated-style interruptions had a similar effect as immediate-style interruptions [59]. In our analysis, we included warnings using both interruption styles (e.g., warning markers as examples for negotiated-style interruption and pop-ups for immediate-style interruptions) and found that our participants' views on the interruption styles reflected the findings of the above studies.

Figure 1: Warning markers yellow and blue

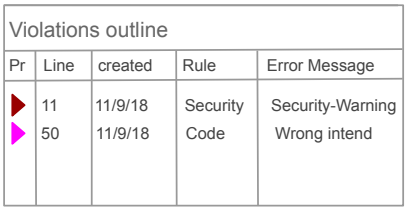


Figure 2: Warning view

3 METHODOLOGY

Our study consists of two parts. First, we conducted a GT study based on Charmaz [21] with theoretical sampling [26] to get a broad overview of the problem space. We followed up this qualitative work with a quantitative survey to test the developed themes and our explanatory theory [25].

To design and test the semi-structured interview guideline for the GT study, Dillman's [30] three-stage pretesting process was followed. First, to decide which types of warnings to use, related work on security bugs was reviewed, and the findings were discussed with eight colleagues from the field of computer science. Second, the guideline was reviewed by a professional developer using a think-aloud approach. Third, a pilot study with two computer science students was conducted to test the study design and to iron out ambiguities. Through this process, some ambiguities were found and eliminated, and wording was improved. The final guideline can be found in supplementary material.

All 26 semi-structured face-to-face interviews were conducted by one researcher and lasted between 30-40 minutes each. In addition to the interviews, we conducted one focus group with seven academic researchers using the same guideline. This was done in order to ensure data reliability and validity by using multiple data-collection methods (methodological triangulation) [28, 29, 39, 62]. The focus group was moderated by the same researcher who conducted the interviews.

The participants were asked which IDEs they had experience with and in which areas of software development they had worked in the past or were still working in. Then, the participants were shown different types of security warnings, which are introduced in the following section. Follow-up questions were asked by the interviewer to clarify different statements the interviewee expressed. All interviews and the focus group session were audio-recorded and transcribed afterwards.

3.1 Types of Warnings

We created a generic picture of an IDE in which we could illustrate different types of warnings. The following warning types were illustrated: (1) underlying code with a yellow line and showing a

Figure 3: Compiler warning

warning marker at the left of the line of code (see Figure 1). This is a common warning marker used by many IDEs. We added one variation of this using (2) blue as a new color for security warnings. We also added a lock icon as the warning marker. (3) We illustrated a *warning view* inspired by static analyzer plugins, such as FindBugs [36], PMD [57], and SonarQube [61] (see Figure 2). This view contains both security and non-security related bugs as is done by these tools. (4) A variation of (3) in which the view contains solely security warnings. (5) We illustrated a security warning as part of a compiler output (see Figure 3). (6) We included a *pop-up warning* since they are commonly used for warning end users even though we have not seen them for security issues within the IDE yet (see Figure 4a). We included two more pop-up warning variants. (7) A pop-up which warns the developer if she is about to commit code that contains security issues (see Figure 4b) and (8) a variant of (7) in which there is an option to create a ticket to delegate the warning to a colleague. The 8th warning type was added as part of the GT approach based on feedback from participants from a security start-up. All warning visualizations were printed out on A4 paper and spread out on a table for the participants to view. Since a number of warnings contain only slight variations, all warnings were explained to participants before feedback was elicited. The warning variations (4) and (8) as well as the interview protocol can be seen in the supplementary material.

3.2 Recruitment

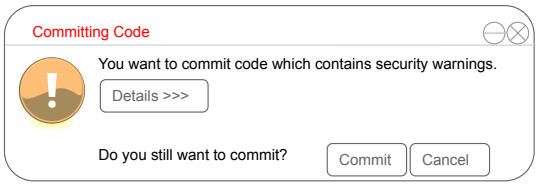
Qualitative Study. The theoretical sampling approach [26] was used to recruit the participants. This means that after coding each interview, new participants were sampled either with similar or different criteria depending on whether categories and themes needed to be confirmed or extended.

We started the GT process with computer science students sampled from our university, through the mailing lists of several different lectures (both security and non-security lectures were sampled). The student participants were compensated with €15 and had a median programming experience of 3.5 years (Md = 3.5, μ = 4.25, σ = 3.72). After we "defined and tentatively conceptualized relevant ideas that indicate[d] areas to probe with more data" [19, p. 107]), we moved on to recruit professional software developers with more programming experience to develop and refine categories. By using our industry contacts, professionals with a median of 10 years of programming experience (μ = 11.85, σ = 8.54) were recruited from different companies including start-ups, government institutions

Gender	Male: 26	Female: 6	Other: 1
Age	min: 19, max: 61	μ: 33, Md: 30	σ: 10.68
University Degree	Yes: 28	No: 4	NA: 1
Main Occupation	Professional developer: 14, Academic researcher: 7	Graduate student: 9	Undergraduate student: 3
Nationality	German: 22, Indian: 3	Bangladeshi: 2, Turkish, Macedonian: 1 each	NA: 4
General Development Experience [years]	min: 1, max: 30	μ: 9, Md: 5	σ: 7.9

Table 1: Qualitative Analysis: Demographics for *n=33*

(a) Pop-up warning while coding



(b) On committing

Figure 4: Pop-up warnings

and financial institutions. Professional developers were compensated with €30.[1] During the GT process we also conducted a focus group with academic researchers from diversified fields of computer science from our institution. The researchers donated their time to the study but were offered cake and refreshments during the study.

Demographics Qual. The demographics of our participants can be found in Table 1. The participants' nationalities included the following: German, Indian, Bangladeshi, Turkish, and Macedonian (with the majority of the participants being German). The sample included more males than females (male: 26, female: 6, other: 1) and a wide range of ages (19 - 61 years, μ = 33, Md = 30, σ = 10.68), as well as a wide range of software development experience (min = 1 year, max = 30 years, μ = 9 years, Md = 5 years, σ = 7.9). The majority of the participants (28/33) had a university degree. Nine participants were graduate students in computer science; 3 were undergraduate students. Furthermore, 14 participants were professional developers (4 start-up, 4 financial sector, 5 public sector, 1 freelance developer), and 7 were academic researchers. We numbered our participants based on where we recruited them. In the following sections, S1-S12 denote students, A1-A7 indicate academic researchers, D1-D14 indicate professional developers. Further information on the participants' development fields, experiences with IDEs and programming languages can be found in the supplementary material.

Quantitative Study. The recruitment of a reasonable number of professional software developers for quantitative research studies is challenging [5, 9, 49, 52]. Thus, we used multiple channels for recruitment. We recruited developers by using (1) a professional recruitment service offered by Qualtrics [2], (2) our database of software developers who previously took part in studies, (3) our professional and industry contacts and (4) XING [3]. Qualtrics calculated that the filling out the survey should take 15 minutes and charged us €43 per participant[2] for recruitment and data collection. We requested that they recruit up to 50 developers for us. Participants recruited via the channels (2), (3) and (4) were compensated with a €20 Amazon gift card.

Since compared to end-user surveys the compensation for our survey was fairly high, we saw the risk of non-developers signing up for our study and faking it and thus, compromising our dataset. To counter this problem, we designed a small programming pre-test with the aim of filtering out non-developers. The test was a multiple choice question with a code snippet where participants needed to understand that the program printed out "hello world" backwards. The question had 6 options of which 2 contained backwards text. We tested the suitability of the question by having 56 MTurk [1] participants and 12 computer science colleagues solve it. Additionally, we asked if participants had programming experience. Of the 56 MTurk, 31 claimed to have programming experience of whom 5 picked the correct answer. Of the remaining 25 MTurkers who stated that they had no programming experience, none picked the right answer. Of the 12 computer science colleagues only two picked incorrectly. The two colleagues and one MTurker who stated to have programming experience picked the incorrect backwards answer. We decided to remove the second backwards option since we did not want developers to spend more time than necessary with the discriminator. The final programming question can be found in the supplementary material with the rest of the survey. All participants were shown the test, independent of our recruitment channels. Only if participants were able to solve the pre-test, we invited them to take part in our online survey. As is standard practice, we also added an attention check question [48] to our survey to filter out careless respondents.

Demographics Quant. Through the different channels, we recruited a total of 176 participants (129 through Qualtrics and 47 through other channels). Of the 129 respondents from Qualtrics, 96 did not pass the programming test, 5 failed our attention check question, and a further 4 were discarded for quality issues. Since a high number of Qualtrics participants failed the programming test, we manually inspected all the remaining participants. We conducted sanity checks to sort out participants with obviously false results,

[1]Two participants rejected the compensation and wanted it to be used for further research.

[2]We asked Qualtrics how much they pass on to the developers but we did not get a number. Their statement was: "Our participants are incentivized through an arrange of different point systems (Airmiles, amazon vouchers, etc). We wouldn't be able to disclose the sheer amount of points though, as we have NDAs with all our Panel Partners."

Gender	Male: 40	Female: 9	Prefer not to say: 1
Age	min: 24, max: 63	μ: 33, Md: 36	σ: 8.9
Teamsize	min: 1, max: 50	μ: 9.9, Md: 6	σ: 9.6
General Development Experience	min: 2, max: 30	μ: 12.96, Md: 11	σ: 8.21
Country of work	German: 25	USA: 18, UK: 4	Europe: 3
Occupation	Software developer: 44, Researcher: 2	IT Manager, IT Security consultant: 1 each	IT Security researcher, Software security architect: 1 each

Table 2: Quantitative Analysis: Demographics for *n=50*

e.g., if their indicated years of employment were higher than their age. Two participants failed the sanity checks and were discarded from our analysis. Finally, we had 22 valid answers of Qualtrics participants.

Of the 47 participants recruited through other channels than Qualtrics, 4 failed the programming test. Thirteen participants passed the programming test, but did not proceed with the online survey and 2 further participants did not pass our attention check. Finally, we had 28 valid responses.

The data reported herein is from the 50 valid responses recruited through Qualtrics and other channels. Average survey completion time was 21.77 minutes (Md = 17.75, σ = 11.30). The demographics of our participants are shown in Table 2. Participants are currently working in development in Germany (n = 25, 50%), the US (n = 18, 36%), the UK (n = 4, 8%) or in Europe (n = 3, 6%). The sample included 40 males and 9 females (other = 1) and a wide range of ages (24 - 63 years, μ = 8.9, Md = 36, σ = 8.9), as well as a wide range of software development experience (min = 2 year, max = 30 years, μ = 12.96 years, Md = 11 years, σ = 8.21). Forty-two of the 50 participants had a university degree. Forty-four participants indicated to work as a software developer, 2 as a researcher and 4 in other IT fields. On average the team size of participants was 9.9 and ranged from 1-50 (Md = 6, σ = 9.6). More information on participants' demographics is available in the supplementary material.

3.3 Evaluation Methodology

Qualitative Study. The interview and focus group transcriptions were evaluated independently and iteratively by two researchers using the *constructivist grounded theory method* of Charmaz [16, 19, 20, 22]. This method is used to establish a theory that is grounded in qualitative data by coding interview transcripts in a repeating process while taking memos. The method assumes that researchers cannot be completely unbiased or without opinions on their research topics: "Rather than being a tabula rasa, constructionists advocate recognizing prior knowledge and theoretical preconceptions and subjecting them to rigorous scrutiny" [20]. Since our understanding of the warnings was developed through literature research and discussions with colleagues, we considered the constructivist approach as an appropriate method to evaluate our data.

Based on GT, our codes emerged inductively as themes were identified during the coding process [40]. After coding each transcription, the codes were updated and the guideline was adapted according to the results. We used theoretical sampling [26] to establish new insights about our codes, categories and themes. Data reliability, validity, and saturation were ensured through *triangulation* via the application of multiple external analysis methods, such as having more than one researcher exploring the phenomenon (*investigator triangulation*) and using multiple data-collection methods as interviews and focus groups (*methodological triangulation*) [28, 29, 39, 62]. Since the focus group was used as a complimentary data acquisition mechanism, we evaluated it accordingly to the interviews by applying the coding techniques of GT methodology [19, 50, 51, 56]. In the following, we present an example for the GT process applied in our study:

Code: After each interview, we coded the interview transcripts and extracted different codes. For example, we extracted the code *functionality first, security second* from the following statement: "*When the code is not running, I will first try to fix the code before addressing security warnings... so before saving my work, let me go through all the warnings, not while I'm coding*" (D5). Another code example for the code *if critical, code is not released* is as follows: "*If there are security relevant findings in the code, they are displayed and if they are critical, the code is not released.*" (D13).

Category: The code examples presented above suggested the category *functionality vs. security*.

Theme: Based on our memos we prepared during coding, codes and categories, we extracted different themes [41]. For example, the above example demonstrated the presence of the theme *when to address security (action)*. Other theme examples presented in our interviews were *how and when to display security warnings, team, and code review*.

Theoretical sampling: We started our initial sampling with students. After we constructed conceptualized categories from data, we sampled to develop these categories, i.e. to obtain more data that helped us to explicate the categories. For example, while full-time students stated that they would consider security after their code is running without errors, student assistants of a security start-up perceived both functionality and security of code as equally important. Thus, the category *functionality vs. security* required to be refined and we continued sampling developers from a start-up company with security focus.

Theoretical saturation: Saturation was achieved when no new themes, theoretical insights and properties of categories emerged. For example, we perceived the category *functionality vs. security* as saturated when we were able to explain it in a "theoretically sufficient" [19] way, i.e. the idea of *functionality first, security second* was not tied to students or experience but to company culture and thus the organization has a huge influence on when developers address security.

Theory: We studied "how–and [...] why–participants construct meanings and actions in specific situations" [19]. As suggested by Charmaz [19], we used the idea of theoretical sorting to develop a diagram out of our memos, themes and

categories offering an initial analytic frame (see supplementary material). Considering this diagram, memos, categories, themes, and the experience with participants, we constructed a resulting theory, which has to be interpreted with respect to the fact that "both researchers and research participants interpret meanings and actions" [19].

Due to the fact, that GT themes and theories evolve continuously throughout the study, we will only present the final themes and theory. The full GT process including themes and the codebook with examples from the interviews can be found in the supplementary material.

In order to obtain the inter-coder agreement, we followed the approach of [18]. The coding schemes were compared, and inter-coder agreement was calculated using Cohen's kappa coefficient (κ) [24]. The agreement measured 0.81 (a value above 0.75 is considered to be a high level of coding agreement [38]).

Quantitative Study. Due to the distribution of our data we used non-parametric tests. We used Friedman's ANOVA to compare the preference score among the warning types. This omni-bus test was followed by pairwise Wilcoxon Rank sum tests with Bonferroni-Holm correction for multiple testing. Further, we used the Pearson correlation test in order to examine correlations between two variables. We chose the common significance level of $\alpha = 0.05$. We used Bonferroni-Holm corrections for all tests concerning the same dependent variable. The corrected p-values are denoted with *cor-p*.

4 ETHICS

The institutional review board of our university approved our project. Participants of our qualitative study were provided with a consent form outlining the scope of the study, the data use and retention policies. We also complied the General Data Protection Regulation (GDPR) regulations. The participants were informed that they could withdraw their data during or after the study without any consequences, as well as the practices used to process and store their data. The consent form was handed out before the interview, and the participants were asked whether they had any additional questions. All of the participants received a copy of the consent form. Participants of our quantitative study were provided the same consent form (adapted for online studies) at the beginning of our survey and had to consent before they could proceed with answering the questions. They were also asked to download the consent form for their own use.

5 LIMITATIONS

This section describes the limitations of our study, which must be taken into account when interpreting the results.

Grounded theory and theoretical sampling is not aimed at creating a representative sample and thus, the qualitative part of our study can make no claims with respect to generalizability. To get a broader view of the developer population, we used several recruiting measures to gather developers for our survey. Due to the difficulty of recruiting developers, we offered a higher payment than is typical for end-user survey, which could temp non-developers to take part and lie about their occupation. To minimize the potential for cheating we included a programming knowledge test to remove non-programmers from our survey. A surprisingly high number (96 of 129) of Qualtrics participants failed this programming test. This made us concerned that the remaining 33 Qualtrics participants might have selected the right answer by chance. However, in our MTurk counter-sample all participants who selected the correct answer also stated that they could program. Since the MTurk participants had no incentive to lie, we feel confident that the majority of correct Qualtrics answers were made by real developers. We nonetheless manually double checked all remaining participants from all samples to check for inconsistencies and did not find anything out of the ordinary and thus, continued with our evaluation. However, since other developer studies have been done using the Qualtrics service e.g., [9], one of the lessons learned from our study is that it is absolutely necessary to include these kinds of test-questions.

Finally, interviews and surveys rely on self-reported data and thus, are subject to self-reporting bias. Therefore, the results of our paper should only be seen as the first step in this research agenda. They are meant to be used to inform the design of further studies in which the participants interact with the different warnings.

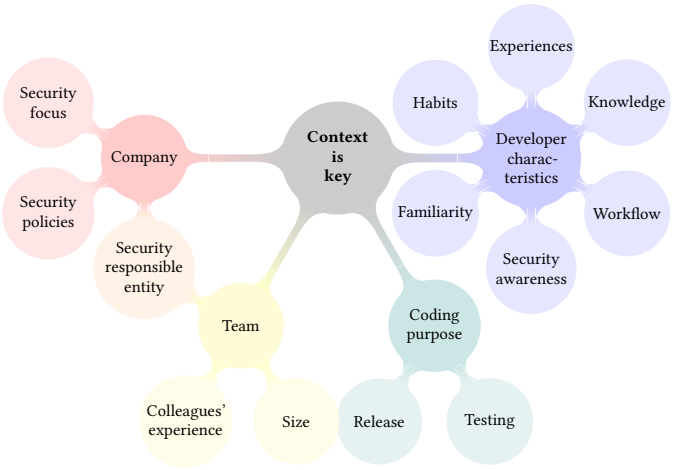


Figure 5: Context aspects to consider for security warning design.

6 RESULTS OF QUALITATIVE ANALYSIS

Through evaluation of the interviews, we found that different factors can influence whether, how, and when developers would like to be warned about security issues. As suggested by qualitative research experts [25], we present an explanatory theory that is grounded in the data collected from our participants:

> **Context is the key to designing and presenting security warnings to developers. It plays a major role in determining how developers will react to security warnings. We found that the developer's stated preferences are based on the coding purpose, developers' characteristics, team, and organization context.**

Figure 5 summarizes the relevant aspects of our key theory. We found that there are a lot of aspects to consider when designing

warnings for developers. First, we found that developers' characteristics play an influential role in security warning preferences. Developers have different levels of knowledge and experience with software development and security; therefore, they require varying levels of support. Moreover, developers cannot stay up to date on every security issue (D1), nor can they ensure that their work is constantly free of errors, and sometimes they might lack security awareness altogether (S1, S5, A4). Additionally, their familiarity with concepts, habits, and workflows can vary, which implies that diverse warning approaches may be necessary.

Moreover, the purpose of developing an application can differ from project to project. For test projects, warnings could be less intrusive or turned off completely. However, measures will be required to ensure that projects with existing security issues are not released by mistake while the security warnings are deactivated (because of remaining in a test mode).

Finally, the team context is important to consider. For example, further requirements emerge if delegation of security issues occurs or a peer review process is possible. On the one hand, more experienced colleagues or a person responsible for security issues could be helpful during the code review process. On the other hand, the participants remarked that it is difficult to work on unfamiliar projects or snippets of code. Furthermore, if nobody feels responsible for delegated security issues, they will remain unsolved. Team size is an especially important factor affecting familiarity with code and feeling responsible for security issues that arise in other developers' work.

Based on the GT approach, we separate our findings into three phases: (1) the display phase; (2) the action phase; and (3) the code-review phase. The context is relevant for all the three phases. The following sections provide more insights on the three phases.

6.1 Phase One: How to Display Warnings

We found *customizability* for security warnings to be a desired feature: "*I need the option to configure everything*" (D8) and "*I think it depends on habits. I don't think there is one IDE to fit everyone*" (A4). In particular, preferences on the different types of security warnings, color and marking code passages differed between developers. Some developers (S5, D9) liked the current yellow warning color, while others favored a specific color for security (S1-S4, S6-S8, S10-S12, D1, D3, D4). S2, S8, D2, and D4 even wanted red as warning color, although red is currently used for syntax errors. Furthermore, A4 proposed looking for different preference profiles and suggesting them to developers. A detailed analysis of the different types of warnings can be found in the following sections.

Warning marker. All of our participants had experiences with the standard warning marker, which appears in yellow. However, some participants (S2, D1, D2, D4) noted that they tended to ignore them. The yellow marker allows code to compile and could, therefore, be regarded as less important. Furthermore, D13 reported that it is difficult to find markers in a large code base, and there is not enough space to display the warning message properly.

By contrast, D9 stated that markers are in-line and can be found directly in the source code without the necessity for other views or windows. Therefore, the warning marker was often favored by our participants. A number of participants (S2-S4, S6, S7, S10, S11, D10) stated that a warning marker with an individual color for security (e.g., blue) is more visible and more specific because security would be highlighted. The start-up participants especially liked the idea of an extra symbol to indicate a security issue (e.g., a lock next to the line). In contrast, D9 and S5 liked the warning marker in general but did not care about the color.

Warning and security view. Most IDEs allow plugins to add sub-windows to the IDE and have the option to switch to dedicated views with a set of windows for a specific purpose, such as debugging or security testing. However, a number of participants (S1, S6, S8, S9, D2, D6, D9, D10, D14) found that the user interfaces of IDEs are already overloaded: "*There is lots of other stuff [...]. And sometimes we don't see those lower left or right corners*" (S6). A1, A4, and D1 also said it was necessary to familiarize themselves with new plugins because they were not accustomed to looking at the corners of the screens where the warnings are positioned. This assumption is supported by the results of Wogalter et al. [67], who found previously that the location of warnings is crucial for warning design.

Developers (D7, D8, D11, D14) who previously worked with plugins, liked them because they provide summaries of important security issues, which creates a more flexible workflow. Therefore, plugins and additional views can give developers the convenience of looking at warnings whenever they have the time to check their code for issues.

Compiler warnings. Our participants had mixed feelings about compiler warnings. Some participants (S4, D1, D3, D6, S12) found that compiler warnings could be easily overlooked because the compiler can produce a lot of information, which can hide warnings.

By contrast, other participants (S1, S2, S9, S11, D4) had positive feelings about compiler warnings and even stated that they could be one of their favorite warning approaches. For example, S2 said that for some developers, compiler warnings could be useful and comfortable if they check the output regularly. Other participants (S2, S9) liked compiler warnings because they do not distract from the programming process. This allowed developers to resolve the problem on their own schedule.

Pop-ups. The majority of our participants did not like the idea of pop-ups because they were considered distracting, annoying (e.g., S1-S3, S5-S12, A1-A4, D1-D9, D11-D14), and disrupted the programming and workflow (e.g., A1, D3, D8). In addition, "*visual staining*" with different pop-ups on the IDE interface was a problem for D9, meaning that pop-ups are visually distracting and facilitate frustration instead of compliance. Since use cases can always differ, false positives could be additionally annoying if they are displayed through pop-ups.

However, while participants with more programming experience had negative opinions about pop-ups, participants with less programming experience (1-4 years of programming experience) favored pop-ups. Some students (S2-S4, S6, S10) found that in very important cases, distracting pop-ups could be useful to catch the attention of the programmer. S6 explained, "*Pop-up warnings could be there for a very important purpose*" so that "*the user is actually forced to solve the issue*" (S4). Additionally, some of the experienced participants (e.g., A1-A7, D8) admitted that pop-ups could be useful in certain situations, such as when a developer initiates an action like committing code.

Warning on committing. The vast majority of our participants found warnings on committing to be one of the best, or even their favorite warning approach which we will discuss below.

6.2 Phase One: When to Display Warnings

Most of our participants wanted to get warnings at two distinct times, once during programming and once before committing.

Many liked the committing warning because it is the most visible and noticeable (S1, S4, A2, D1, D3). This type of warning also fits perfectly into their workflow because they are required to look for bugs and security issues before committing (S8, A2, D1, D3, D5, D8). They especially liked the idea of "*having an additional stop sign that says: Are you really sure?*" (A2) to highlight security issues they might have forgotten or missed (S6, A1, A2, D1, D3, D8).

However, there was also vocal support for being shown warnings as soon as something was done wrong (S1, S2, S4, S6, S7, S11, D1, D4-D7). However, participants also stated that they would like to have warnings shown when they find time to deal with the issues because, as D4 stated, "*Quality costs time. That's the way things are*". This of course is a lot harder for a computer program to judge, but it is an interesting research goal. In other words, workflow needs to be respected when warning developers:

> "*On one hand, you want to bother people at the right time. On the other hand, you don't want to prevent people from working*" (A2).

A common problem most participants in Johnson et al.'s [46] study faced with analysis tools, was the inability to temporarily ignore or suppress certain warnings. Our participants had different opinions on whether it should be possible to postpone the display of warnings with security focus. Several participants (S1, S8, S11, D1, D6, D7) said that disabling security warnings should not be possible because security is of high importance and forgetting about security issues can have fatal consequences (S1, D13, D14). Therefore, S11 and A1 liked the idea of handling warnings as errors, which would force them to fix the issues, while D4 requested blocking commits and merge requests if security warnings are not fixed.

However, several of our participants (S10-S12, A1-A2, D1, D4, D12, D14) pointed out the importance of identifying why a specific application is being developed (code purpose). If developers are programming an application for test purposes, disabling warnings should be possible even within a security context. If an application is released to the public, the programmers definitely have to receive security warnings. Especially, developers who are not aware of security issues should be warned before they submit projects in the "test mode" (A1, A4). Still, the majority (S4, S9, S10, S12, D2-D4, D6, D8, D9, D11-D14) indicated that ignoring warnings should be possible because false positives can occur and the tools need to stay flexible.

6.3 Phase Two - Action

While the characteristics and purpose of an application influence how and when developers want security warnings to be displayed, the organization can affect the perceived trade-off between functionality and security. Consequently, the organization can determine when developers act and address security issues.[3]

We found that participants working in different contexts had different attitudes about how and when to approach security issues in their software. First, most of the student participants (S1-S4, S6, S7-S10) said they would consider security issues after their program was functional. This observation was also found in a password storage study conducted with computer science students by Naiakshina et al. [53]. Usually, computer science students are programming in a university context where they are not required to consider security issues outside security lectures and thus, often instead concentrate on submitting functional solutions in the first place. It seems that the institutional context can affect when a computer science student chooses to address security.

Moreover, we found that professional developers from companies without a strong emphasis on security (A2, A4, A5, D6, D7, D10) weigh up on organization requirements on when and how to deal with security issues:

> "*As soon as [the program] has to pass through quality control [...], we have to fix the security issues*" (A5).

Time restrictions could lead developers to neglect security (D7). In addition, participants from the public sector (D6-D10) reported that general security guidelines existed in their organization, but they were not project specific and did not provide enough technical guidance (D8, D9). Additionally, nobody in the organization was responsible for verifying whether these security guidelines were met (D8, D10). Still, the participants from the public sector (D6-D10) referred to various static analyzer tools that are used in their organization, after their programs were finished.

By contrast, companies that place a high emphasis on security required developers to ensure the security of their software. D11, D12, and D13 noted that they took extra time to address security issues because of policies that needed to be fulfilled before submitting their projects. Accordingly, developers from security-focused companies reported addressing security issues as soon as they arose (D1-D4) or at the end of the development process (D11-D14). Consequently, these participants preferred viewing warnings immediately and on committing with an automated security ticket.

6.4 Phase Three: Code-Review

S12 stated that in large open-source projects, it is difficult to deal with warnings because he is not familiar with the code created by other developers. Therefore, in further interviews, we asked whether our participants would like to be shown the security issues of other colleagues. We found that the team context can influence the code-review process during the software development life cycle. Thus, perceptions of the various types of warnings differed from team to team. In order to involve more experienced colleagues in the review process, participants from the start-up company that emphasized security, requested an option to create a security ticket automatically when committing. We also found that this idea was preferred by developers who were employed by security-focused companies (D1-D4, D11-D14). For example, D3, D4, and S5 indicated

[3]We have to note, that developers' personal traits might also affect their preference for certain employers and vice versa.

that they liked to see the warnings of team members, especially when code was merged with unsolved security issues. In their organization, a maintainer reviews merge-requests, and they thought it was important for the code reviewer to see their unsolved warnings as well.

However, developers working for companies without an emphasis on security and those working alone or in a big team said that such an option could lead to unsolved issues (D8-D10). D10 provided an example of someone who could take advantage of such an situation: "*Reporting frees me of responsibility! [...] I have reported [security issue], so I will never have to work on it again!*" The participants pointed out that project organization (D8 and D14) and the team size (D9) should also be considered. D9 argued that with 5 or 6 developers on a team, it could be possible to realize an effective delegation process. However, he also identified problems that could arise:

> "*We have frequent changes among the staff, and hire external developers [...] so it doesn't make sense here. [...] If you have a security team it would make sense*" (D9).

Delegating a warning can create a situation where nobody feels responsible for solving the issue because responsibility is not clearly defined (D9, D10, D12).

7 QUANTITATIVE STUDY

Based on our qualitative findings, we wanted to test our main hypothesis, that no one warning type dominates all others.

7.1 Survey Design & Warning Preference Score

To gather further insights and test our hypothesis, we designed and ran an online survey. The full survey can be found in the supplemental material.

Since we wanted to keep the survey below 20 minutes, we decided to combine the similar warning types from the qualitative study. Our qualitative findings revealed that the related types of warnings *warning view* and *security view* were similar evaluated by our participants. Therefore, they were only shown one warning and simply asked whether they wish to have a global or a security specific warning view. We also showed three different warning markers in yellow, blue and red in a single image and let participants additionally choose their color of preference. To make the warnings easier to see on a computer screen and particularly mobile devices, we cropped out the IDE leaving only the actual warning in the image. Participants were shown the different types of warnings in a randomized order and were asked to rate the following 6 statements on a 7-point Likert-Scale [37] for each type of warning:

- S1 I would like to be informed about security issues in my code with this type of warning.
- S2 I would be quickly annoyed by this type of warning.
- S3 It would be easy to overlook this type of warning.
- S4 IDEs already have too many warnings of this type.
- S5 I am familiar with this type of warning.
- S6 This type of warning would fit into my workflow.

The statements were extracted based on participants' views of different warnings types from our qualitative research. Further, we asked questions about the different context aspects which we established in our qualitative analysis like organization policies and participants' characteristics.

In order to compare the warning preference of our participants, we calculated a *warning preference score* for each type of warning. We included 4 of the 6 statements to the score, which could be evaluated as a positive (S_1 and S_6) or a negative (S_2 and S_3) attitude towards a security warning. S4 and S5 were not included in the preference score since they cannot be assessed as either positive or negative statements. For instance, not being familiar with one type does not make the type automatically more or less liked. We calculated the score of at most 24 points as follows: ((S_1 + (7-S_2) + (7-S_3) + S_6) - 2).

7.2 Results of Quantitative Analysis

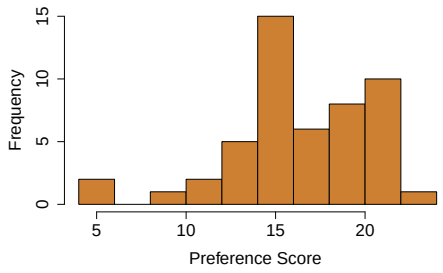
Our analysis of the quantitative study is structured in the following way. First, we provide an analysis on how developers want to be warned comprising our hypothesis. Then, we give insights on when developers want to be shown the warnings, at which time they want to deal with the warnings and how the different types of warnings fit in their workflow. Finally, we discuss the committing warning with security ticket based on the reported team sizes.

One Size Does Not Fit All. Looking at the mean values of the warning preference scores for our warning types, we can provide a ranking. The minimum score was 0 and the maximum score was 24.

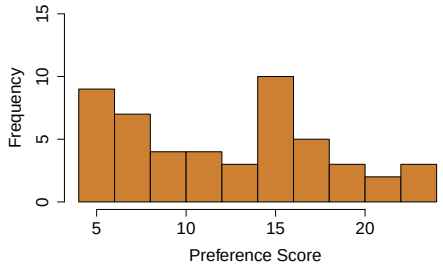
(1) Warning marker (mean = 16.9, σ= 4.06)
(2) Warning view (mean = 16.6, σ= 3.86)
(3) Compiler warning (mean = 14.7,σ= 4.91)
(4) Warning on committing (mean = 14.68, σ= 4.7)
(5) Warning on committing with security ticket (mean = 14.46, σ=4.97)
(6) Pop-up warning (mean = 12.8, σ= 5.5)

For the warning marker participants favored different colors: 22% chose blue, 22% favored red, 18% preferred orange 14% fancied violet and 14% yellow, 4% picked pink, 1 participant chose green, 1 grey and 1 answered "configurable no fixed color". One participant noted "Please consider color blinds or different color schemes when designing. A few of my teammates are color blind and IDEs are like hell for them."

No one warning type dominates all other. In order to assess our main hypothesis that different warning types appeal to different developers, we used a Friedman's ANOVA omnibus test to examine the differences of the means of our 6 warning types (p <0.000*, χ^2_f = 481.49). Afterwards, we calculated post-hoc pairwise Wilcoxon Rank Sum tests and corrected them with the Bonferroni-Holm correction. After the correction, the results show that the best (warning marker) and worst (pop-up warning) rated warning types have significant different scores (*cor-p* = 0.003*) as well as pop-up compared to warning view (*cor-p* = 0.007*). Figures 6a and 6b display the frequencies for the preference scores for warning markers and pop-up warnings. Figure 6a shows that while the majority of participants rated the warning marker with a higher preference score, however, there are some low ratings as well. For the pop-up warnings (Fig. 6b) the opinions split into a positive and negative camp. And despite pop-up warnings having the lowest overall score, 11 out of 50 participants ranked it higher than the marker warnings, which



(a) Warning marker: warning preference score



(b) Pop-up warning: warning preference score

Figure 6: Histograms of warning preference scores (0-24)

was the most popular warning type. But the easiest way to see that no one or even two warning types would suit all developers is shown in Figure 7. Here we plotted the preference score ranking of 10 random participants for each type of warning. Each spike in the graph indicates a preference jump within one participant and the spread of point shows that all warning types have low, high and medium scores. This shows that the preferences our participants reported cannot be met by any tool that only offers one or even two warning types.

Years of programming experience and warning preferences. While our qualitative results suggested that the more experience developers had, the more positive they were about the different warning types (except for pop-ups where the reverse was true), our tests did not reveal any significant correlation for any of the warning types.

We tested whether there is a correlation between the years of programming experiences and the preference for warnings, but found no correlation. Warning preferences could depend on developers' experience level but we found no evidence in our sample for a negative or positive correlation.

Personal security focus is correlated with higher preference score for at least one warning type. We asked our participants to rate their personal focus between security and functionality using a slider with the mid-point indicating equal focus. We coded a security leaning focus as a positive value for *security focus*. We found that the self-reported security focus followed a normal distribution over our participants. We selected the preference score of the favorite warning type of each participant and tested whether this score

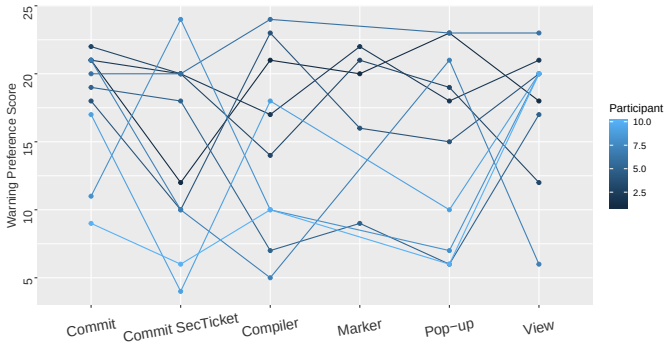


Figure 7: Preference scores for 10 random participants

correlated with the security focus. We used the maximum preference score instead of a combined score since it could be that one participant favors one type of warning in particular while disliking all other types of warnings. We found a small positive correlation (r = 0.343) between the self-reported security focus and the preference score for the most favored warning type (*cor-p* = 0.01*, confidence interval (CI) = [0.07, 0.56]). Participants who reported to have a higher security focus, were more likely to rate one or more warning types with a higher preference score. We can make no claim as to the causal relationship between these two variable, however, we believe it is worth exploring whether positive experiences with warnings could increase the security focus of developers.

While coding - right away	42%
While coding - after completing a function	44%
While coding - in regular intervals	20%
Before running	56%
Before committing	64%
Before release	40%
On demand (e.g. by clicking a button or opening a view)	60%
Never	0 %

Table 3: Time preference of warnings

When to Display Warnings. Table 3 summarizes the time points when our participants preferred to be warned. Multiple answers were possible and as above it is clear that no one time is the clear winner. It is likely that configurability would be beneficial.

Snoozing warnings. Figure 8 shows participants' views on how they would like to be able to ignore warnings. As can be seen most features are fairly well balanced, again showing the heterogeneity of developers' wishes. The least popular option by overall weight is turning off security warnings entirely, but even there almost 50% expressed a positive view.

Action - Secure coding policies and responsibility. Almost one half of our participants (46%) reported to have no security coding policies from their organization. We found that the security focus appeared to be normally distributed, showing that developers have different security focuses. Further, the security focus of the organizations

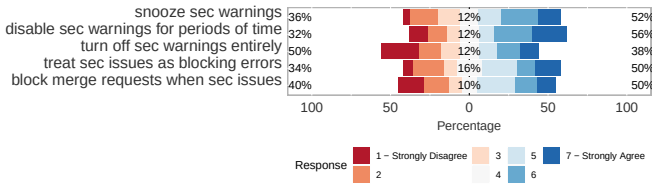


Figure 8: Handling security warnings

was also reported to vary across our participants. While the security focus of the organization is indeed important, 39 (78%) reported to feel responsible for security bugs.

Workflow. Confirming Johnson et al.'s [46] qualitative finding that workflow concerns matter, we found that the statement that a warning type suits a developers workflow (S_6, Section 7.1) has a strong positive correlation with participants stating that they would like to be warned with this type of warning (S_1,Section 7.1) (p<0.000*, CI = [0.73, 0.81]) with r = 0.77. This emphasizes that workflow compatibility is of upmost importance.

Code-Review. We found that opinions on committing with automatic security tickets vary a lot. 36% said that committing with a security ticket would not fit into their workflow. 58% reported that they would write a security ticket for themselves while 17/50 reported that the security ticket should be for another colleague responsible for solving security issues. 60% said that they do not have a team or person responsible for code security in their organization. However, 82% reported to have a colleague whom they could ask in cases of security issues on an informal base.

8 DISCUSSION AND RECOMMENDATIONS

In this work we explored developers' perceptions about different warning types and times at which warnings should be shown to warn them of security issues with their code. Our results are very clear. The current practice of many tools to offer a single type of warning always shown at the same time, does not offer the best user experience for a large percentage of developers. While previous work has suggested more flexibility in warning configuration within a single warning type, our work suggests that this flexibility needs to go further and span multiple warning types. Our work also confirms Johnson et al.'s qualitative work [46] and shows a high level of correlation between developers perceived compatibility of warning types with their development workflows and their rating of these warnings.

However, our work also cautions against some suggestions made in previous work. Wurster and van Oorschot [68] have suggested handling security warnings as blocking errors to enhance security. This is also an idea expressed by some of our interviewees. However, the majority of our participants favored configuration that allowed warnings to be to snoozed or disabled (e.g., during testing).

In the following we summarize recommendation we have drawn from our data.

Warning type. Tools should offer developers the choice of which warning type or types they would like to receive for security issues, e.g., marker, view, pop-up, commit, etc.

Warning time, snooze and delegation. Developers should be able to specify when they want to deal with security warnings and have the ability to snooze or delegate them. This is inline with Johnson et al.'s [46] qualitative results.

Context is key: Workflow Taking the workflow of developers into account is highly important but can be complex in some cases. While it is easy to show warnings before developers commit code, other wishes of developers are less straightforward to implement and require further research. For instance, a simple time based snooze is most probably not an ideal solution. A more intelligent snooze functionality that can take into account the current programming context and workflow of the developer holds more potential.

Context is key: Environment Team composition, role within the team, company policies and personal knowledge level all play important roles in how developers want to be warned. Current warning designs try to create the single best warning. Our research suggest that warnings will need different form, content and workflow integration based on the environmental context.

9 CONCLUSION

We conducted a qualitative grounded theory based study with 33 participants (26 face-to-face interviews and a focus group with 7 participants) from different fields of software development and evaluated their opinions on diverse security warning approaches. We developed the theory that context is the key to designing and presenting security warnings to developers. It plays a major role in determining how developers will react to security warnings. We found that the developers' stated preferences are based on the coding purpose, developers' characteristics, team, and organization context. We followed this up with an online survey with 50 professional developers to develop a more detailed understand and develop recommendations for warning design. The quantitative results confirm that context and in particular workflow play a major role and that developers have very different wishes where it comes to warning interaction. Our take away message is that one size does not fit all and it would be beneficial if tools not only give developers more configuration options within a single warning type but offers a selection of warning types and times.

We see two interesting avenues for future research. Firstly, the data we gathered in this study is self-reported. We plan to run several lab and field studies to test the effects configuration options. In particular, we are interested to see real world interaction with security pop-up warnings and to tie in warning content with warning types. Furthermore, we think it is an interesting field of research to find the right time to warn developers without clear cut preferences. Using machine learning it might be possible to detect times when developers would be particularly receptive to warnings or when they are in the flow and should not be disturbed.

ACKNOWLEDGMENTS

This work was partially funded by the ERC Grant 678341: Frontiers of Usable Security.

REFERENCES

[1] [n.d.]. Amazon Mechanical Turk (MTurk). https://www.mturk.com/ Accessed: August 2019.

[2] [n.d.]. Qualtrics. https://www.qualtrics.com Accessed: August 2019.

[3] [n.d.]. XING. https://www.xing.com/ Accessed: August 2019.

[4] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2017. Comparing the usability of cryptographic apis. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, IEEE, Piscataway, NJ, USA, 154–171.

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

[6] Devdatta Akhawe and Adrienne Porter Felt. 2013. Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. USENIX, Washington, D.C., 257–272. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/akhawe

[7] Hazim Almuhimedi, Adrienne Porter Felt, Robert W Reeder, and Sunny Consolvo. 2014. Your reputation precedes you: History, reputation, and the chrome malware warning. In *10th Symposium On Usable Privacy and Security ({SOUPS} 2014)*. 113–128.

[8] Ammar Amran, Zarul Fitri Zaaba, and Manmeet Kaur Mahinderjit Singh. 2018. Habituation effects in computer security warning. *Information Security Journal: A Global Perspective* 27, 4 (2018), 192–204.

[9] Hala Assal and Sonia Chiasson. 2019. 'Think Secure from the Beginning': A Survey with Software Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 289, 13 pages. https://doi.org/10.1145/3290605.3300519

[10] Brian P Bailey, Joseph A Konstan, and John V Carlis. 2000. Measuring the effects of interruptions on task performance in the user interface. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, Vol. 2. IEEE, 757–762.

[11] Titus Barik, Denae Ford, Emerson Murphy-Hill, and Chris Parnin. 2018. How Should Compilers Explain Problems to Developers?. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. ACM, New York, NY, USA, 633–643. https://doi.org/10.1145/3236024.3236040

[12] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. 2017. Do Developers Read Compiler Error Messages?. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, Piscataway, NJ, USA, 575–585. https://doi.org/10.1109/ICSE.2017.59

[13] Cristian Bravo-Lillo, Lorrie Cranor, Saranga Komanduri, Stuart Schechter, and Manya Sleeper. 2014. Harder to Ignore? Revisiting Pop-Up Fatigue and Approaches to Prevent It. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*. USENIX Association, Menlo Park, CA, 105–111. https://www.usenix.org/conference/soups2014/proceedings/presentation/bravo-lillo

[14] Cristian Bravo-Lillo, Lorrie Faith Cranor, Julie Downs, and Saranga Komanduri. 2011. Bridging the gap in computer security warnings: A mental model approach. *IEEE Security & Privacy* 9, 2 (2011), 18–26.

[15] Cristian Bravo-Lillo, Saranga Komanduri, Lorrie Faith Cranor, Robert W. Reeder, Manya Sleeper, Julie Downs, and Stuart Schechter. 2013. Your Attention Please: Designing Security-decision UIs to Make Genuine Risks Harder to Ignore. In *Proceedings of the Ninth Symposium on Usable Privacy and Security (SOUPS '13)*. ACM, New York, NY, USA, Article 6, 12 pages. https://doi.org/10.1145/2501604.2501610

[16] Antony Bryant and Kathy Charmaz. 2007. *The Sage handbook of grounded theory*. Sage.

[17] Ross Buck, Mohammad Khan, Michael Fagan, and Emil Coman. 2018. The user affective experience scale: A measure of emotions anticipated in response to pop-up computer warnings. *International Journal of Human–Computer Interaction* 34, 1 (2018), 25–34.

[18] John L Campbell, Charles Quincy, Jordan Osserman, and Ove K Pedersen. 2013. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological Methods & Research* 42, 3 (2013), 294–320.

[19] Kathy Charmaz. 2006. *Constructing grounded theory: A practical guide through qualitative analysis*. sage.

[20] Kathy Charmaz. 2008. Constructionism and the grounded theory method. *Handbook of constructionist research* 1 (2008), 397–412.

[21] Kathy Charmaz. 2014. *Constructing grounded theory*. sage.

[22] Kathy Charmaz and Linda M McMullen. 2011. *Five ways of doing qualitative analysis: Phenomenological psychology, grounded theory, discourse analysis, narrative research, and intuitive inquiry*. Guilford Press.

[23] Maria Christakis and Christian Bird. 2016. What developers want and need from program analysis: an empirical study. In *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, IEEE, Piscataway, NJ, USA, 332–343.

[24] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[25] Juliet Corbin, Anselm Strauss, and Anselm L Strauss. 2014. *Basics of qualitative research*. sage.

[26] Imelda T Coyne. 1997. Sampling in qualitative research. Purposeful and theoretical sampling; merging or clear boundaries? *Journal of advanced nursing* 26, 3 (1997), 623–630.

[27] David M DeJoy, Kenneth R Laughery, and Michael S Wogalter. 1999. *Warnings and risk communication*. Taylor & Francis.

[28] NK Denzin. 1970. The Research Act in Sociology (London, Croom Helm). *Denzin The Research Act in Sociology 1970* (1970).

[29] Norman K Denzin. 2017. *The research act: A theoretical introduction to sociological methods*. Routledge.

[30] Don A Dillman. 2011. *Mail and Internet surveys: The tailored design method–2007 Update with new Internet, visual, and mixed-mode guide*. John Wiley & Sons.

[31] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. 2008. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1065–1074.

[32] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*. ACM, New York, NY, USA, 50–61. https://doi.org/10.1145/2382196.2382205

[33] Adrienne Porter Felt, Alex Ainslie, Robert W Reeder, Sunny Consolvo, Somas Thyagaraja, Alan Bettes, Helen Harris, and Jeff Grimes. 2015. Improving SSL warnings: Comprehension and adherence. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. ACM, 2893–2902.

[34] Adrienne Porter Felt, Serge Egelman, and David Wagner. 2012. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 33–44.

[35] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. 2012. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*. ACM, 3.

[36] FindBugs. 2018. FindBugs - Find Bugs in Java Programs. http://findbugs.sourceforge.net/. Accessed: 28-01-19.

[37] Kraig Finstad. 2010. Response interpolation and scale sensitivity: Evidence against 5-point scales. *Journal of Usability Studies* 5, 3 (2010), 104–110.

[38] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. 2013. *Statistical methods for rates and proportions*. John Wiley & Sons.

[39] Patricia Fusch, Gene E Fusch, and Lawrence R Ness. 2018. Denzin's paradigm shift: Revisiting triangulation in qualitative research. *Journal of Social Change* 10, 1 (2018), 2.

[40] Barney G Glaser and Anselm L Strauss. 2017. *Discovery of grounded theory: Strategies for qualitative research*. Routledge.

[41] G Glaser Barney and L Strauss Anselm. 1967. The discovery of grounded theory: strategies for qualitative research. *New York, Adline de Gruyter* (1967).

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

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

[44] Scott Hudson, James Fogarty, Christopher Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny Lee, and Jie Yang. 2003. Predicting human interruptibility with sensors: a Wizard of Oz feasibility study. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 257–264.

[45] Brittany Johnson, Rahul Pandita, Justin Smith, Denae Ford, Sarah Elder, Emerson Murphy-Hill, Sarah Heckman, and Caitlin Sadowski. 2016. A Cross-tool Communication Study on Program Analysis Tool Notifications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. ACM, New York, NY, USA, 73–84. https://doi.org/10.1145/2950290.2950304

[46] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE, Piscataway, NJ, USA, 672–681. http://dl.acm.org/citation.cfm?id=2486788.2486877

[47] Stefan Krüger, Sarah Nadi, Michael Reif, Karim Ali, Mira Mezini, Eric Bodden, Florian Göpfert, Felix Günther, Christian Weinert, Daniel Demmler, et al. 2017. CogniCrypt: supporting developers in using cryptography. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*.

IEEE Press, IEEE, Piscataway, NJ, USA, 931–936.

[48] Franki YH Kung, Navio Kwok, and Douglas J Brown. 2018. Are attention check questions a threat to scale validity? *Applied Psychology* 67, 2 (2018), 264–283.

[49] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*. ACM, New York, NY, USA, 492–501. https://doi.org/10.1145/1134285.1134355

[50] Manuel Mora, Ovsei Gelman, Annette Steenkamp, and Mahesh S Raisinghani. 2012. *Research methodologies, innovations and philosophies in software systems engineering and information systems*. IGI Global Hershey, PA.

[51] David L Morgan. 1996. Focus groups. *Annual review of sociology* 22, 1 (1996), 129–152.

[52] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. 2019. "If You Want, I Can Store the Encrypted Password": A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 140, 12 pages. https://doi.org/10.1145/3290605.3300370

[53] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why Do Developers Get Password Storage Wrong?: A Qualitative Usability Study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 311–328. https://doi.org/10.1145/3133956.3134082

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

[55] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. A Stitch in Time: Supporting Android Developers in WritingSecure Code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 1065–1077. https://doi.org/10.1145/3133956.3133977

[56] Richard A Powell and Helen M Single. 1996. Focus groups. *International journal for quality in health care* 8, 5 (1996), 499–504.

[57] PMD Open Source Project. 2018. PMD- An extensible cross-language static code analyzer. https://pmd.github.io/. Accessed: 28-01-19.

[58] Robert W. Reeder, Adrienne Porter Felt, Sunny Consolvo, Nathan Malkin, Christopher Thompson, and Serge Egelman. 2018. An Experience Sampling Study of User Reactions to Browser Warnings in the Field. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 512, 13 pages. https://doi.org/10.1145/3173574.3174086

[59] TJ Robertson, Joseph Lawrance, and Margaret Burnett. 2006. Impact of high-intensity negotiated-style interruptions on end-user debugging. *Journal of Visual Languages & Computing* 17, 2 (2006), 187–202.

[60] TJ Robertson, Shrinu Prabhakararao, Margaret Burnett, Curtis Cook, Joseph R Ruthruff, Laura Beckwith, and Amit Phalgune. 2004. Impact of interruption style on end-user debugging. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 287–294.

[61] SonarSource SA. 2018. SonarQube - The leading product for Continuous Code Quality. https://www.sonarqube.org/. Accessed: 28-01-19.

[62] Clive Seale. 1999. Quality in qualitative research. *Qualitative inquiry* 5, 4 (1999), 465–478.

[63] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bei-Tseng Chu, and Heather Richter. 2018. How developers diagnose potential security vulnerabilities with a static analysis tool. *IEEE Transactions on Software Engineering* (2018).

[64] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. 2009. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *Proceedings of the 18th Conference on USENIX Security Symposium (SSYM'09)*. USENIX Association, Berkeley, CA, USA, 399–416. http://dl.acm.org/citation.cfm?id=1855768.1855793

[65] Joel Weinberger and Adrienne Porter Felt. 2016. A week to remember: The impact of browser warning storage policies. In *Symposium on Usable Privacy and Security. https://www. usenix. org/system/files/conference/soups2016/soups2016-paper-weinberger. pdf.*

[66] Tara Whalen and Kori M Inkpen. 2005. Gathering evidence: use of visual security cues in web browsers. In *Proceedings of Graphics Interface 2005*. Canadian Human-Computer Communications Society, 137–144.

[67] Michael S Wogalter, Vincent C Conzola, and Tonya L Smith-Jackson. 2002. Research-based guidelines for warning design and evaluation. *Applied Ergonomics* 33, 3 (2002), 219 – 230. https://doi.org/10.1016/S0003-6870(02)00009-1 Fundamental Reviews in Applied Ergonomics 2002.

[68] Glenn Wurster and Paul C van Oorschot. 2009. The developer is the enemy. In *Proceedings of the 2008 New Security Paradigms Workshop*. ACM, ACM, New York, NY, USA, 89–97.