
How to Conduct Security Studies with Software Developers

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
Anastasia Danilova
aus
Sankt Petersburg, Russische Föderation

Bonn, Dezember 2021

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn.

Erstgutachter: Prof. Dr. Matthew Smith
Zweitgutachterin: Dr. Katharina Krombholz

Tag der Promotion: 02.05.2022
Erscheinungsjahr: 2022

Abstract

As our everyday life depends on software and its security, studies with software developers become ever more important. Especially in the field of software security, each mistake can have major ramifications as the data of millions of users can be compromised. Motivated by multiple security breaches caused by issues during software development, more studies with software developers have to be conducted to investigate and help software developers create and test secure software.

Thus, the design of security studies with software developers is a crucial task for researchers in Human Computer Interaction (HCI). While ample research has been conducted with end-users, there has been less investigation involving software developers. As with to end-user studies, different methods can be used to conduct studies with software developers, and the design decisions need to be made carefully and rationally. Since recruiting developers as study participants is difficult and costly, the decisions referring to study design have an increased value in the research process. Therefore, it is important to consider different methods before deciding on any one of them. These decisions can be made during the various stages of a study. Deception in the task design or the study announcement can be used to increase ecological validity. Further, screening the online survey participants with software developers can help improve the data quality of the results. Designing study tasks for developer studies opens lots of possibilities. Besides writing code, reviewing code can also yield different results. Finally, it is to explore whether qualitative and quantitative analysis show different results or whether qualitative analysis might suffice for key aspects of the research topic is another concern of this work.

In order to gain more insights into methodological design decisions, this thesis considers different approaches that have been used to conduct security studies with software developers. Five developer studies in different fields are conducted to investigate their methodological and usability-related implications. The findings of each study are analyzed at the primary level as well as a meta-level. While the primary level deals with the implications for that specific security topic, the meta level deals with methodological insights and findings.

In addition to the methodological contributions, the five studies offer insights into various security topics for software developers. The results indicate that secure password storage is still a challenge for software developers. Furthermore, security warnings during software development can be displayed in different ways, and there is no solution for all software developers; rather, the warning design should respect the facets of the software development life cycle.

Based on the methodical findings, this thesis offers recommendations for conducting security studies with software developers, in particular with regards to screening, announcing, setting up the task and analyzing the results in security studies with software developers.

The results suggest that screening for programmers is useful and necessary to increase data quality in online surveys with software developers. Further, deception in the study announcement does not yield different results regarding security awareness and scores. Code-reviewing appears to be a useful and promising method for software developer studies, and the results in terms of security awareness are similar to code-writing. Finally, similar key results in the qualitative analysis of our study suggest that qualitative analysis can help point out key results in software developer studies. Based on these results, recommendations are offered on how to conduct security studies with software developers.

Acknowledgments

First of all, I would like to thank my supervisor Prof. Dr. Matthew Smith for his trust and constant advice throughout this time. His dedication inspired me to push myself harder.

I'm extremely thankful for my great friend Alena Naiakshina for her constant support. Whenever I felt down, you were there to pick me up again and I knew you would always have my back. Thank you for the wonderful time and the impeccable teamwork. I couldn't wish for a better friend and colleague.

Further, I'd like to thank all my colleagues who I was able to work with, in particular Eva Gerlitz, Stephan Plöger, Christian Tiefenau, Maximilian Häring, Mischa Meier, Klaus Tulbure, and Marco Herzog. Also, I'd like to thank my co-authors, Stefan Horstmann, Johanna Deuter, and Anna Rasgauski for their great work. Without your help, this work would not have been possible.

I would like to thank my wonderful family, especially my mother Viktoria and sister Ksenia for always supporting and believing in me. Finally, I'd like to thank my partner Ramy for always being there for me and encouraging me throughout this process.

Table of Contents

- 1. Introduction** **1**
 - 1.1. Thesis Structure 3

- 2. Related Work on Developer Studies** **6**
 - 2.1. Developer Security Studies 6
 - 2.1.1 Studies with Freelancers 6
 - 2.1.2 Warning Studies 7
 - 2.2. Password Storage Studies 9
 - 2.3. Code-Reviewing Studies 10
 - 2.4. Ecological Validity 11
 - 2.5. Recruitment and Screening 12
 - 2.5.1 Recruitment Strategies 12
 - 2.5.2 Identifying Programming Skill 13

- 3. Analysis: Security Warnings for Software Developers** **15**
 - 3.1. Motivation 15
 - 3.2. Methodology 16
 - 3.2.1 Types of Warnings 17
 - 3.2.2 Recruitment 18
 - 3.2.3 Evaluation Methodology 20
 - 3.3. Ethics 22
 - 3.4. Limitations 22
 - 3.5. Results of Qualitative Analysis 23
 - 3.5.1 Phase One: How to Display Warnings 24
 - 3.5.2 Phase One: When to Display Warnings 26
 - 3.5.3 Phase Two - Action 26
 - 3.5.4 Phase Three: Code-Review 27
 - 3.6. Quantitative Study 28
 - 3.6.1 Survey Design & Warning Preference Score 28
 - 3.6.2 Results of Quantitative Analysis 29
 - 3.7. Discussion and Recommendations 32
 - 3.8. Summary 33

- 4. Data Quality: Screening Questions for Online Surveys with Programmers** **35**
 - 4.1. Motivation 35
 - 4.2. Methodology 37
 - 4.2.1 Instrument Requirements 37
 - 4.2.2 Survey 37
 - 4.2.3 Statistical Testing 39
 - 4.2.4 Participants 39
 - 4.3. Ethics 41
 - 4.4. Limitations 42

4.5. Results	42
4.5.1 Effectiveness	42
4.5.2 Efficiency	45
4.6. Testing the Instrument	45
4.6.1 Non-adversarial Test Group	46
4.6.2 Attack Scenario	47
4.7. Discussion and Recommendations	49
4.8. Summary	50
5. Data Quality: Time Limits in Screener Questions with Programmers	52
5.1. Motivation	52
5.1.1 Baseline Study	53
5.2. Methodology	54
5.2.1 Tasks	54
5.2.2 Conditions	55
5.2.3 Participants	56
5.2.3.1 Programmers: Computer Science Students	57
5.2.3.2 Non-programmers: Clickworker	57
5.2.4 Statistical Analysis	57
5.2.5 Ethics	58
5.2.6 Limitations	58
5.3. Results	58
5.3.1 Programmers	59
5.3.1.1 Ensuring Comparability with the Baseline Study	59
5.3.1.2 Comparison Between Groups	59
5.3.1.3 Self Evaluation	60
5.3.1.4 Pressure	61
5.3.2 Non-programmers- Timed Attack Scenario	62
5.3.2.1 Comparison to the Baseline Attack Scenario	62
5.3.2.2 Comparison to Programmers	63
5.3.2.3 Aid Used	64
5.4. Discussion and Recommendations	64
5.4.1 Screener Tasks	65
5.4.2 Time Constraints	65
5.4.3 Setup of the Instrument	65
5.5. Summary	66
6. Deception: Study Announcement in a Password-Storage Study	68
6.1. Motivation	68
6.2. Methodology	70
6.2.1 Study Design Changes	70
6.2.2 Pilot Study	72
6.2.3 Participants	72

6.2.4	Evaluation	73
6.3.	Limitations	74
6.4.	Ethics	74
6.5.	Results	74
6.5.1	Security	74
6.5.2	Prompting effect (H-P1)	76
6.5.3	Java and Password Storage Experience (H-G1, H-G2)	76
6.5.4	Framework (H-F1)	76
6.5.5	Security Guidelines (NIST and OWASP)	77
6.5.6	Sample Comparison	79
6.6.	Discussion	81
6.7.	Summary	82
7.	Task Design: Comparison of Code-Reviewing and Code-Writing in Developer Studies	84
7.1.	Motivation	84
7.2.	Methodology	85
7.2.1	Survey	86
7.2.2	Code Snippets	87
7.2.3	Participants	87
7.2.4	Evaluation	88
7.2.4.1	Security	88
7.2.4.2	Qualitative	89
7.2.4.3	Quantitative	89
7.2.5	Ethics	89
7.2.6	Limitations	90
7.3.	Results	90
7.3.1	Qualitative Analysis	90
7.3.1.1	Participants' Review Criteria	91
7.3.1.2	Found Password Storage Issue	91
7.3.1.3	Security Score	92
7.3.1.4	Distraction Tasks	92
7.3.1.5	Ready for Release?	93
7.3.1.6	Participants' Definition of Hashing and Salting	93
7.3.1.7	Security Responsibility	93
7.3.2	Quantitative Analysis	95
7.3.2.1	Effect of Prompting on the Word Count	95
7.3.2.2	Effect of Prompting on Finding the Password Storage Issue	96
7.3.2.3	Effect of Experience on Finding the Password Storage Issue	96
7.3.2.4	Effect of Different Insecure Code Snippets on Finding the Password Storage Issue	96
7.3.2.5	Time for Security	97
7.4.	Discussion	97
7.5.	Summary	99

8. Conclusions	101
A. Appendix: Analysis: Security Warnings for Software Developers	104
A.1. Structure	104
A.2. Qualitative Study	104
A.2.1 Semi-structured Guideline	104
A.2.2 Demographic Questions	105
A.2.3 Demographics of Participants	106
A.2.4 Grounded Theory Analysis	106
A.2.4.1 Codebook	106
A.2.4.2 Themes	107
A.3. Quantitative Study	110
A.3.1 Types of Warnings	110
A.3.2 Invitation	110
A.3.3 Survey	111
A.3.4 Demographics of Participants	117
A.3.5 Warning Rating	119
B. Appendix: Data Quality: Screening Questions for Online Surveys with Programmers	134
B.1. Structure	134
B.2. Survey	134
B.3. Participants' Demographics	139
B.3.1 Country of Residence	139
B.3.2 Main Occupation	139
B.4. Results	140
B.4.1 Effectiveness	140
B.4.2 Efficiency	140
B.4.3 Attack Scenario	140
B.5. Timing thresholds	140
C. Appendix: Data Quality: Time Limits in Screener Questions	147
C.1. Questions	147
C.1.1 Attention Check Question	147
C.1.2 Demographic Questions	147
C.1.3 Screener Questions	147
C.1.4 Students Introduction	153
C.1.5 Attack Scenario Introduction	153
D. Appendix: Deception: Study Announcement in a Password-Storage Study	155
D.1. Security Requests	155
D.2. Security Scale	155
D.3. Hypotheses	157
D.4. Summary of Statistical Analysis	157

D.5. Playbook	157
D.5.1 Study Announcement and Study Offer	157
D.5.2 Deadline	158
D.5.3 Password-related Questions	158
D.5.4 General Questions	158
D.5.5 Receiving the Solution and Survey Request	159
D.5.6 Exit Communication	159
D.5.7 Review	159
E. Appendix: Task Design: Code-Reviewing and Code-Writing	160
E.1. Survey	160
E.2. Playbook	162
E.3. Code Snippets	164
E.4. Evaluation of Participants' Code Reviews	164
E.5. Participants' Review Criteria	164
E.6. Found Password Storage Issue	164
Bibliography	168

1. Introduction

As the work and lives of millions of computer users depend on software and its security, studies conducted with software developers are crucial to help prevent security issues and improve the usability of software developer tools. Mistakes made by software developers can result in enormous data losses for end users as multiple security breaches in the wild have demonstrated in the past [88, 178, 83, 109, 68, 74]. The necessity to prevent those scenarios from happening again is unquestionable. Still, prior research has showed that mistakes can repeat themselves because software developers still have issues with secure programming, and they demonstrate misconceptions of various security concepts like encryption, password storage or Transport Layer Security (TLS) [129, 88, 178, 83, 109, 12, 117, 161, 11, 14, 114, 192, 45, 63, 133, 131, 149, 23].

Thus, more studies with software developers must be conducted in order to investigate and support experts in developing secure software. Green and Smith [98] raised the issue and suggested different approaches to improve the usability of cryptographic Application Programming Interfaces (API). Similar to end-user usable security [15], more knowledge on the topic of security studies with software developers is needed. This results in the call for more studies of this specific nature. While this topic has been thoroughly discussed for end-user security and usability studies, more study is needed into security studies with software developers.

In their research agenda, Acar et al. [12] proposed different methods to conduct security studies with software developers. In particular, they suggested conducting more research to clarify whether Computer Science (CS) students would yield similar results as professional developers in security studies with software developers. We [132, 133, 131, 130] investigated whether freelancers and CS students could be regarded as proxies for study participants for developers. Moreover, the question of whether an online study would produce different results than a lab study was discussed. The results showed no significant differences between the studies.

Ecological validity [152] tackles the question of whether study results are caused by the study itself or would also hold true in the real world. Thus, ecological validity has to be considered in several main design decisions made in security studies with software developers. Ecological validity can involve several seemingly small nuances which could lead to varying results. Example questions are: Should the participants be asked to solve a task securely? Should the task be announced as a scientific study, or should deception be used, disguising it as a real job? Comparison studies are needed to answer these questions and formulate more recommendations on *how* to conduct security studies with software developers.

Deception and data quality are two main aspects to be considered in this thesis. *Deception* can be employed as a strategy to increase the ecological validity of security studies with software developers, but it opens up new ethical issues [111, 110], and it can have negative implications for both participants and researchers. For example, it could result in distrust or a perceived lack of transparency in future studies. Therefore, deception needs to be utilized wisely, and as rarely as possible. *Data quality* is crucial for security studies with software developers since it ensures that the data are accurate.

Comparison studies are needed to study security studies with software developers. We [130, 131, 132] investigated whether deception in task design would be necessary to investigate security-related behavior and security awareness. The results showed that prompting for security in the task yields more secure solutions.

Table 1.1.: Methodological topics in security studies with software developers

<i>Category</i>	<i>Topic</i>	<i>Study</i>
Deception	Study announcement	[58]
Deception	Prompting vs. non-prompting	[130, 131, 132]
Data Quality	Screening for programming skill	[60]
Data Quality	Screening for programming skill with time limits	[62]
Task Design	Code-writing vs. code-reviewing	[61]
Study methods	Qualitative vs. quantitative analysis	[59]
Study participants	Freelancers	[132]
Study participants	Company developers	[133]
Study participants	Computer Science students	[130, 131]

Rows in bold denote the contributions of this thesis. The grey rows were already investigated in related work.

This thesis presents studies that all refer to the main issue of the best ways to conduct security studies with software developers. Several questions surrounding security studies with software developers remain unanswered. The main goal of this thesis is to investigate how to conduct security studies with software developers with regard to data quality and ecological validity. Various aspects were discussed during the different stages of conducting a study. The four aspects I investigated in detail are depicted in Table 1.1. The main goal was to provide recommendations on how to conduct security studies with software developers through an attempt to provide answers to the following research questions:

- **Deception during study announcement:** Should researchers announce security studies with software developers as real world assignments or as scientific studies?
- **Data quality by screening participants:** How can researchers ensure data quality by screening the study participants effectively for online surveys by ensuring a minimal programming skill? Could the screening be improved by imposing time limits?
- **Code-writing vs. code-reviewing in task design:** Is code-writing as a task the only way to obtain insights into participants' knowledge and experience of a secure programming topic in security studies with software developers? Could code-reviewing show similar tendencies in security studies with software developers, and how would this influence the security awareness of the participants?
- **Qualitative vs. quantitative analysis:** Is a quantitative analysis always required for conducting security studies with software developers or can a qualitative analysis offer valuable insights?

In a quest for further insights, this thesis presents and discusses different approaches tested to conduct security studies with software developers. Several security aspects are discussed in more detail, and this demonstrates the misconceptions that developers often have regarding topics such as password storage. Moreover, it aims to provide insights based on our studies to help other researchers conducting security studies with software developers. To give indications for the research questions presented above, I have designed and conducted several comparison studies with software developers. This thesis ultimately offers methodological recommendations for future research.

First, I present a study on the screening process to improve data quality by verifying a minimum level of participants' programming skills without them writing code. Recruiting participants for developer studies is a challenging task [131] due to a lack of time and local unavailability of participants, and the unaffordability of hourly rates [11, 14, 114, 192, 161, 12, 117]. Therefore, researchers can recruit online via platforms like Qualtrics [7] or clickworker [1] that can help with the recruitment of developers. To prevent the study noise by participants lacking programming skills, different screener questions were developed and tested with different samples of non-programmers and programmers. As a result, a number of questions, with and without time limits, to effectively filter out a number of non-programmers are recommended for future use. Further, a follow-up study investigates time limits for the recommended questions and meets the need for short and effective screener questions.

Furthermore, I investigate whether announcing a study as an actual job, and not as a study, would change the results in security studies with software developers. Thus, I present a study with freelance developers which explored this difference and demonstrated that the results remained similar under both conditions. While we [132] studied freelancers without announcing the password storage study as a scientific study, the replication without deception yielded the same effect sizes and directions.

Further, I conducted a study with freelance developers to test the code-reviewing methodology and come to an initial understanding of the differences using code-reviewing and code-writing as study tasks. This study dives into the subject of different methodologies for similar research questions. Even with code-reviewing instead of code-writing, prompting for security in the task description showed similar indications in terms of the security awareness of participants.

Finally, one study deals with evaluating the results of security studies with software developers. As recruiting software developers in large numbers can be challenging [131], researchers often rely on a qualitative analysis with a smaller number. To compare the trends in the findings, I compared the qualitative and quantitative parts of the two studies with a similar topic. While the questions differed, the results of the qualitative study showed similar trends to those of the quantitative study.

1.1. Thesis Structure

The findings of this work have been accepted by five peer-reviewed conference papers with me as first author. Each paper is presented in one chapter. The thesis is structured as follows:

Chapter 1 describes the motivation and research questions of this thesis, the contribution and thesis structure.

Chapter 2 summarizes the related work for this thesis. The chapter's contents were originally part of the publications [58, 59, 60, 61, 62].

Chapter 3 describes a qualitative and quantitative study on security warning preferences. The contents of this chapter were published as parts of the publication "One Size Does Not Fit All: A Grounded Theory and Online Survey Study of Developer Preferences for Security Warning Types", Anastasia Danilova, Alena Naiakshina, and Matthew Smith, presented at the *42nd International Conference on Software Engineering 2020 (ICSE 2020)* [59].

Chapter 4 features the first study on establishing a screening questionnaire. The contents of this chapter were published as parts of the publication "Do you really code? Designing and Evaluating Screening Questions for Online Surveys with Programmers" by Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith, presented at the *43rd International Conference on Software Engineering (ICSE 2021)* [60].

Chapter 5 describes time limits in screening questions. The contents of this chapter are part of the publication "Testing Time Limits in Screener Questions for Online Surveys with Programmers" by Anastasia Danilova, Stefan Horstmann, Matthew Smith, and Alena Naiakshina, and is going to be presented at the *44th International Conference on Software Engineering (ICSE 2022)* [62].

Chapter 6 contains the replication study of a password-storage study. The contents of this chapter were published as parts of the publication "Replication: On the Ecological Validity of Online Security Developer Studies: Exploring Deception in a Password-Storage Study with Freelancers" by Anastasia Danilova, Alena Naiakshina, Johanna Deuter, and Matthew Smith, presented at the *16th USENIX Symposium on Usable Privacy and Security (SOUPS 2020)* [58].

Chapter 7 describes a qualitative analysis of password-storage study with a code-reviewing as methodology. The contents of this chapter were published as parts of the publication "Code Reviewing as Methodology for Online Security Studies with Developers - A Case Study with Freelancers on Password Storage" by Anastasia Danilova, Alena Naiakshina, Anna Rasgauski, and Matthew Smith, presented at the *17th USENIX Symposium on Usable Privacy and Security (SOUPS 2021)* [61].

Chapter 8 offers a summary over the results of the studies from Chapter 3 to 6. Based on the findings, it offers recommendations on conducting security studies with software developers.

Disclaimers at the beginnings of Chapters 3 to 7 provide details on my co-authors' and my contributions for that particular chapter. To emphasize the collaborate work on each paper, instead of "I", the pronoun "we" will be used in this thesis. Only with the help of the co-authors the referenced

work was possible. Therefore, I would like to emphasize my deepest gratitude for their support. Further, all chapters are based on previously published or currently under review work.

2. Related Work on Developer Studies

This chapter summarizes related work on developer security studies. First, it focuses on the primary level of developer studies, in particular the implications for the security topic of each study. After that, the focus lies upon the meta level of developer studies discussing useful methods and considerations for developer studies.

2.1. Developer Security Studies

Balebako et al. [28] conducted semi-structured interviews with 13 app developers. They investigated their view on security- and privacy-related topics. Many participants stated that both were part of their development process but that they were not their top priority. The authors found that security and privacy are positively correlated.

Acar et al. [11] invited 54 Android developers to take part in a lab study comparing different kinds of resources (free choice of resources, Stack Overflow only, official Android documentation only, books only). The participants only using Stack Overflow wrote significantly less secure code than the others, while the ones using only books wrote significantly less functional code than the others.

In a further study [13], Acar et al. compared the usability of five different APIs. They asked 256 GitHub users to solve various tasks concerning symmetric and asymmetric encryption. Twenty percent of the users who solved the task functionally believed to have solved it securely while it was not. The researchers recommended better documentation with secure, easy-to-use code examples.

A study investigating having GitHub members for developer studies was conducted by Acar et al. [14]. 307 participants completed three small python programming tasks and then filled out a survey, all without payment. One of the tasks was to store login credentials in a database. The researchers set out to explore how well GitHub users can replace IT-professionals in developer studies, who often do not have time for studies outside of their working schedules and who often have an hourly rate much higher than that offered for taking part in studies. They found that whether a participant is a professional or a student had no significant effect on the security perception of the participant or on the functionality or security of the solution implemented. Experience did however have an effect: each year of added python experience increased the chance of a secure solution by 5%.

Tahei et al. [170] reviewed literature looking at security from a developer perspective. They found there was generally a lack of research in developer-centered-security, and that security needed to become of higher business value. As found in the previously mentioned studies, security knowledge among software developers is often lacking and security is often a secondary requirement.

2.1.1. Studies with Freelancers

Ahmed and van den Hoven [16] discussed the freelancers' responsibility and ability to cause harm. They pointed out that most freelancers only do exactly what they are asked to do, which can cause security issues with employers who do not have a computer science background—an observation which was also found in Naiakshina et al.'s studies with freelancers [132] and students [130, 131] indicating that this issue cannot be reduced to freelancers only. Another problem Ahmed and van den Hoven found was that freelancers use malicious code from the Internet without testing it to

ensure functionality and security. The researchers wanted to encourage freelancers to accept their responsibility. In our study, we acknowledge these aspects by providing participants password storage standard sources, if they submitted non-secure solutions.

In a further study, Bau et al. [32] compared the websites developed by start-up developers with websites they asked freelancers from Elance.com and Freelancer.com to develop. They wanted to investigate how the employment status, the developer's security knowledge and the programming language influence web application security. Nineteen start-ups and 8 freelance developers were invited and all of them were interviewed and took part in a security quiz. Their analysis showed that the code written by freelancers had more weaknesses than the code written by the start-ups. There was a huge discrepancy between the freelancers' security knowledge and their implementations. Furthermore, they found that code written in PHP had more injection vulnerabilities than code written in other programming languages.

While Bau et al. referred to freelancers as being rather unreliable, in another freelancer study, Yamashita and Moonen [194, 193] conducted a study with 85 freelancers on code smells and acknowledged the flexibility, the access to a wide population, and the low costs of Freelancer.com. Furthermore, in contrast to Bau et al., Naiakshina et al. [132] reported freelancers to be very dependable in their study. Most of the subjects delivered their solutions within the time frame they promised; they were also reliable in their communication and showed a high interest in the study results.

2.1.2. Warning Studies

Barik et al. [30] 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. [31] 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. [64] 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 [189] 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. [94] 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 [51] 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. [135] 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. [162] 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. [107], 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. [108] 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. [27] 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. [101] 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. [147, 148] 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 [147] 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 [148]. 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.

2.2. Password Storage Studies

With regard to password policies, Bonneau and Preibusch [38] analyzed 150 websites which offer free user accounts for various purposes. They found a great variety of security implementations. Websites with few security-critical features had the worst security practices. Other websites storing more sensitive data, such as financial data, implemented better security.

In 2007, Prechelt [143] arranged a contest where teams of web developers took part using different web development platforms (Java EE, Perl, PHP). They all had 30 hours to implement the same requirements for a web-based application. Their results were compared along various factors like usability, functionality and security. They found that PHP was in many aspects “at least as safe” as the other platforms and that it tended to have the smallest within-platform variations. Finifter and Wagner [82] conducted further analysis on the code of [143]. The authors investigated the relation between programming language and its number of vulnerabilities and the frameworks’ support for security features and number of vulnerabilities. They did not find a relation between choice of programming language and application security, but they noticed that the developers almost never made use of the frameworks’ built-in security support, e.g., for password storage. Like Finifter and Wagner, we also investigated whether freelancers would make use of the Spring frameworks’ built-in libraries for secure password storage.

In a further study, Acar et al. [14] invited 307 GitHub users to work on security related tasks (e.g., password storage) in Python and to take part in a survey afterwards. The authors found a positive correlation between performance regarding security and functionality and years of programming experience.

Another password-storage study was conducted by Wijayarathna and Arachchilage [186] with 10 developers. The authors explored usability issues of the Bouncycastle API to provide insights on how to develop, design and improve security/cryptographic APIs. They identified 63 issues in the SCrypt implementation of Bouncycastle.

A qualitative study from Naiakshina et al. [130] in 2017 investigated the password-storage implementation behavior of 20 CS students. The participants were given a scenario where they were told they were working in a team to implement the registration functionality of a social networking platform. Half of the students were prompted beforehand and told to implement a secure solution. None of the non-prompted students implemented a secure solution. As often found with end users, there was also a tendency with developers to offer functionality over security. This study was extended by the authors in 2018, by inviting 20 additional CS students [131]. The authors acknowledged that they had challenges to recruit enough participants for the study. Out of a pool of 1600 CS students at their university, only 40 participated in the study. The exploratory quantitative study supported the findings of the qualitative study. However, CS students stated that they would have implemented a secure solution had the task been for a real company.

To find out whether the previous results were a study artifact, in 2019 Naiakshina et al. [132] conducted a similar study, but this time with 43 freelancers. Hired through Freelancer.com, the participants were asked to implement the registration functionality for a fictitious online sports-picture sharing platform. Here it was also found that prompting has an effect on the security of the end solutions. 15 of the 22 non-prompted, and even 3 of the 21 prompted freelancers received security requests after submitting insecure solutions [132]. Unlike the previous study,

the participants believed they were working for a real company. This did not appear to have an impact on the security of the end solution. The same study was also conducted with 36 professional developers from diverse companies in 2020 [133] and again, prompting had an effect on the security of participants' submissions.

Another study on password-storage was conducted by Wijayarathna et al. [186]. Their study was similar to the one described in the previous papers. To explore the usability of the SCrypt password hashing functionality of Bouncycastle, the authors conducted a 2-hour study with 10 programmers and reported 63 usability issues with the Bouncycastle API. A further study from Wijayarathna et al. [185] investigated how security responsible programmers feel when writing code. The participants were expected to complete four programming tasks including a password storage task. They found that developers know they are responsible for security, but often have a difficult time implementing security measures.

There is a wide variety of studies investigating end-user password creation, password creation rules and their effects and password usage [37, 72, 99, 157, 165, 175, 177, 176, 180, 112, 158, 121, 155]. Although developers play an important role in secure password storage too, rather little work has been done with them. To offer more insights into developers' security behavior, Naiakshina et al. conducted password-storage studies with CS students and freelancers [130, 131, 132].

2.3. Code-Reviewing Studies

Baum et al. [33] looked at code-reviewing practices in 19 firms, 11 of which regularly conduct some kind of code review. They found that some firms use multiple reviewers, particularly when making large changes to code. Aside from being used to improve the quality of the code and find defects, the researchers found that code reviews are also conducted to enable a learning process of the coder. Furthermore, it was found that some firms do not conduct code reviews: developers can often feel attacked by code reviews and need time to adjust to the new method. Bacchelli et al. [26] supported the finding that code reviews are often used as a learning process in firms. They also found that "finding defects" in code was a driving factor for conducting code reviews. In multiple papers the term "finding defects" included security issues, but mostly included other kinds of defects [33, 26, 151].

Using the largely open source Mozilla project, Kononenko et al. [113] studied the quality of code-reviews. 54% of bugs were not found during the code-review. The authors found a positive correlation between reviewing experience and number of bugs found. Coding experience did not have an impact on the number of bugs found. They found larger patches and an increased number of files to be reviewed increased the chance of introducing "buggy" patches. The use of super reviewers, a reviewer who is a highly experienced developer, decreases the probability of introducing patches with bugs in them.

What effect availability bias has on code reviews was investigated by Spadini et al. [164]. Using a browser-based reviewing tool chosen by the researchers, the participants were expected to review a code change. The change contained three errors: two errors were of the same type and were bugs that reviewers do not usually look for, whereas as the third was of another type. Half of the participants were non-primed and received a code change where none of the bugs had been commented on, whereas as the other half were primed and received a code change where one of the errors had been commented on by another reviewer. The researchers found a correlation

between priming the participants and the participants finding the other error of this type. 80% of participants mentioned they were influenced by the comments in the code. The non-primed bug was found by both groups at a similar rate.

A study investigating the optimal number of code-reviewers for a given task was conducted by Edmunson et al. [70]. The participants were web developers all with various backgrounds and security experience. They were presented with and expected to review an existing open-source project, which had some security issues. Edmunson et al. found a correlation between the number of correct vulnerabilities found and the number of false vulnerabilities found. Having more than 15 reviewers did not bring any further improvement. Interestingly, the researchers found a negative correlation between the number of years of experience and the number of correct vulnerabilities found. Our study differs from Edmunson et al.'s study in several aspects. First, while Edmunson et al.'s participants were informed about security issues within the web application they had to review, we did not mention that our code had issues at all. Additionally, half of our participants were advised to consider security for the code review, while the other half were not. Second, in addition to the security issues, we also added general issues within the code, so we were able to see which issues participants are more aware of, if they recognized them at all. Third, we showed participants one simple code snippet in the context of secure password storage, whereas Edmunson et al. investigated Cross-Site Scripting and SQL injection vulnerabilities within an entire web application project.

Most work using code reviewing as a study methodology was conducted in the field of software engineering. It is unclear yet, whether the findings are transferable to a security context. For example, Kononenko et al. [113] found a positive correlation between reviewing experience and number of bugs found, unlike the study from Edmunson et al. [70]. With our study we aim to provide deeper insights into code reviewing as a methodology for developer studies within a security context.

2.4. Ecological Validity

Acar et al.[12] argued that the examination of ecological validity—whether studies reflect the real world—is of great importance. Most work in this field was primarily conducted with end users. For example, Redmiles et al. [145] compared field data with survey reported data about software updates initiated with end users. They found that self-reported data in some cases varied from field data.

Furthermore, Wash et al. [180] compared 134 self-reports of end users' password behavior with their actual behavior and found only a weak correlation of self-reported intentions with reality. In [181], Wash et al. aimed at finding out which security behaviors were accurately self-reported by the end users. For this, they collected survey responses and behavior data from 122 participants. The authors concluded that security self-reports oftentimes do not reflect users' actual behaviors, e.g., if the behavior involves awareness.

Fahl et al. investigated the ecological validity of a password study [75]. They compared the study-observed behavior of 645 participants with their real-life password choices. They conducted an online and laboratory research under priming and non-priming conditions. The authors found that around 20% of the participants behaved differently in the study compared to their real-life

password behavior. They concluded that ecological validity is an important criterion, as it can reveal a high index of the irrelevance of laboratory studies to the real-life behavior.

Simultaneously, Mazurek et al. [122] studied the guessability of passwords used by members of a university in comparison to passwords that were previously collected in experiments or were leaked from low-value accounts. Having the same password policy, the authors found the real university passwords to be more similar to the passwords from research studies than when comparing university passwords to the leak passwords.

In order to increase the ecological validity of developer studies, Stransky et al. [166] designed an online platform which enables developers to participate in a study using their own equipment and allows them to participate from anywhere they would like to. The authors used the platform to conduct two studies and found that participants created code that was equally good as the code created in previous lab studies and noticed that participants were willing to spend more time when working online.

Finally, Naiakshina et al. [130, 131, 132] investigated whether deception task design—prompting participants to secure password storage—would change their security behavior in comparison to non-prompting. The study results showed that prompting has an effect on the security of participants' solutions. We replicated the freelancer study of Naiakshina et al. [132] in order to explore what impacts removing the deception emulating ecological validity might have.

2.5. Recruitment and Screening

In order to provide context for this field of work, we separate our related work into two sections. We first outline developer studies, where recruitment strategies are discussed or chosen in a way that developers are targeted with a high probability. Second, we provide insight into other work that either uses identification and verification mechanisms in the research or investigates how participants could be tested for programming skill.

2.5.1. Recruitment Strategies

Researchers mostly designed the recruitment process in such a way that only software developers were targeted, such as through recruitment of participants on GitHub [186, 94, 14, 97, 13] or on freelance platforms for developers [193, 194, 32, 132]. For example, when investigating the effect of happiness on productivity, Graziotin et al. [97] recruited their participants by contacting software developers on GitHub [91]. GitHub developers were also recruited for a security developer study conducted by Acar et al. [14] to investigate how they perform with regard to security-related tasks. Furthermore, Meyer et al. [124], when conducting a study on job satisfaction and productivity of software developers, directly recruited employees from Microsoft to ensure that all participants indeed work as software developers.

A number of recruitment strategies were compared by Baltes and Diehl in [29]. They recruited survey participants using different approaches: via a personal network, online networks and communities, directly contacting companies, public media, survey advertisement by software engineers, and by using information from GHTorrent [2], which collects data on public GitHub projects. They concluded that the most efficient way to recruit is to use public media and asking software engineers to advertise the survey. Nevertheless, Baltes and Diehl did raise the question as

to whether recruitment strategies like commercial recruiting services or crowdsourcing platforms (e.g., Amazon Mechanical Turk) are suitable for developer recruitment, but did not follow up on it, which we do. Another concern is also raised by Yamashita and Moonen [194], who discussed the recruitment of freelance developers for surveys on the platform Freelancer.com. The authors acknowledged that employers rely on self-reported skills by members of such platforms. They suggested running skill assessment tests to ensure the internal validity of research findings. Our work addresses this issue.

2.5.2. Identifying Programming Skill

Bergersen et al. [35] constructed and validated an instrument for measuring Java programming skill. This was done by evaluating participants' performance on programming tasks. In a study lasting two days, 65 professional developers solved 19 Java programming tasks. Each task had a time limit of between 10 and 45 minutes. The research goal of Bergersen et al. was to construct an instrument to find the best programmers when recruiting for a job or allocating to projects. The task time of 10-45 minutes is too long for our purpose of screening participants before a survey.

Feigenspan et al. [76] conducted an experiment with 125 students and compared the self-reported programming experience with the participants' performance in solving program comprehension tasks. They found a correlation between self-reported experience and performance using a 40 minute survey instrument. In addition to that, the authors highlighted that the software-engineering community is lacking a clear definition of *programming experience*. However, by using this term, researchers often referred to years a participant was programming by using a specific programming language or in general. Interestingly, Bergersen et al. [35] found a correlation between programming experience and skill, which was largest during the first year of experience and disappeared after about four years. While we also used program comprehension tasks in our study, we did not aim to understand how experience correlates with performance; our objective was to find a simple way to exclude participants without programming skill from online developer studies.

In [28], Balebako et al. conducted a study concerning the behavior of smartphone app developers in regard to security and privacy. Potential participants had to fill out a screening survey to qualify for the interviews. This survey included two technical questions to test for knowledge of app development. In addition, the authors verified their findings in a follow-up online survey with participants recruited on various online forums. Again, knowledge and attention check questions were required to be correctly answered by participants. Of the 460 responses, 232 had to be discarded because they did not fulfill the requirements. We contacted Balebako et al. and asked for their used screening questions. Unfortunately, the authors were not able to provide us the exact questions. However, they remembered to have asked for IDEs participants have experience with and apps they developed. After that the authors manually inspected whether the mentioned IDEs or apps exist. The authors acknowledged that especially the IDE question was hard to verify, because of the high amount of IDEs existing "in the wild." Instead of asking for IDEs, we therefore decided to provide participants a list of programming languages including non-existing ones.

In [11], Acar et al. studied the effect of information resources for software development on security by conducting an online survey and a lab study. For the online survey, the researchers sent about 50,000 emails to developers they were able to identify on Google Play. Of the 302

participants who completed the survey, Acar et al. excluded 7 for invalid answers. Additional filters to find non-developer participants were not applied. In the lab study, participants were recruited based on their experience with app development, having either completed a course on Android development or with work experience of at least one year. The participants first had to complete a short programming task to demonstrate their skills. However, following complaints that the task took too much time, they were instead tested with 5 multiple choice questions covering basic Android development knowledge, at least 3 of which needed to be answered correctly. This is a good example showing that programming tasks are not well suited as screening question and that multiple choice questions are more acceptable. We contacted Acar et al. and learned that their screening questions specifically covered Android development knowledge. Since we aimed to identify questions for general programming skill, we did not include them to our instrument. However, we included questions affecting developers beyond Android development, such as error handling.

We [59] investigated the developers' preferences about security warnings in IDEs and tested participants' programming skill with a simple pseudocode multiple choice question. A detailed description of the study can be found in the Chapter 3. To further evaluate the used question, we included it to our question set as Q16.

Assal and Chiasson [25] conducted online surveys to investigate developers' software security processes. A section of their participants were recruited through a paid service provided by Qualtrics. During survey completion, participants were provided different descriptions of software security and were prevented from progressing till they chose the authors' preferred definition of security. Participants initially providing incorrect answers were not excluded from evaluation. The authors wanted to ensure a baseline of security understanding, rather than to test for software security skill. However, participants who provided invalid data or completed the survey too quickly were excluded from evaluation.

As can be seen there is currently no common approach to detecting whether participants have programming skill. Each set of authors come up with their own ideas and no instrument has been tested in any rigorous manner.

3. Analysis: Security Warnings for Software Developers

Disclaimer: The contents of this chapter were published as parts of the publication "One Size Does Not Fit All: A Grounded Theory and Online Survey Study of Developer Preferences for Security Warning Types", presented at the 42nd International Conference on Software Engineering 2020 (ICSE 2020) [59]. This paper was written in cooperation with my co-authors Alena Naiakshina and Matthew Smith. The survey and qualitative guideline were designed by me and discussed by Alena Naiakshina and Matthew Smith. I conducted the qualitative interviews and set up the quantitative survey. Alena Naiakshina assisted during the qualitative evaluation by analyzing the interviews. The statistical analysis of the quantitative data was led and executed by me. Alena Naiakshina, Matthew Smith and I prepared the paper for publication.

3.1. Motivation

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 [19]. 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 [169, 40, 41, 17, 184, 182, 39, 71, 79, 78, 77, 18, 146, 43]. 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 [13, 130, 131, 132] or the misuse of TLS enabling man-in-the middle attacks [74]. Consequently, improving security warnings for developers is a desirable research goal [98].

Many tools can be used by developers to spot potential security issues, such as compilers or static and dynamic code analyzers (e.g., [30, 115, 135]) 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 [94]; however, Barik et al. showed that reading compiler warnings takes a significant amount of effort and that improvements are needed [30, 31].

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 [107].

In this work 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 chapter 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.



Figure 3.1.: Warning markers yellow and blue

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 chapter is organized as follows: First, we discuss our methodology in detail (Section 3.2). Then, we present our explanatory theory in Section 3.5, whereas our quantitative analysis can be found in Section 3.6. In Section 3.7 we discuss the implications of our study and provide recommendations for security warning design for developers.

3.2. Methodology

Our study consists of two parts. First, we conducted a GT study based on Charmaz [48] with theoretical sampling [57] 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 [56].

To design and test the semi-structured interview guideline for the GT study, Dillman's [67] 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 the Appendix A.2.1.

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) [65, 66, 153, 89]. 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.

Violations outline				
Pr	Line	created	Rule	Error Message
▶	11	11/9/18	Security	Security-Warning
▶	50	11/9/18	Code	Wrong intend

Figure 3.2.: Warning view

```

Run:
Security-Warning:
In line 11
You used ...
This can cause ...
Secure-Action:
...
Insecure-Action:
...
Background Details:
[1] ...
INFO: Server started. Service Active.
INFO: Server running.
INFO: Log: Client logged in.

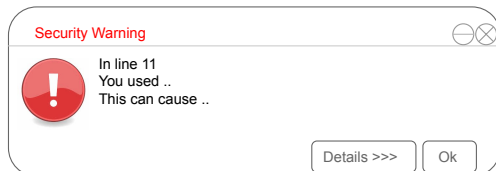
```

Figure 3.3.: Compiler warning

3.2.1. 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 *warning marker* at the left of the line of code (see Figure 3.1). 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 [81], PMD [144], and SonarQube [150] (see Figure 3.2). 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 3.3). (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 3.4a). 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 3.4b) 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 Appendix A.2.

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 3.1.: Qualitative Analysis: Demographics for $n=33$ 

(a) Pop-up warning while coding



(b) On committing

Figure 3.4.: Pop-up warnings

3.2.2. Recruitment

Qualitative Study

The theoretical sampling approach [57] 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” [46, 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 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 3.1. 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

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

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 Appendix A.2.2.

Quantitative Study

The recruitment of a reasonable number of professional software developers for quantitative research studies is challenging [12, 117, 132, 25]. Thus, we used multiple channels for recruitment. We recruited developers by using (1) a professional recruitment service offered by Qualtrics [7], (2) our database of software developers who previously took part in studies, (3) our professional and industry contacts and (4) XING [8]. Qualtrics calculated that the filling out the survey should take 15 minutes and charged us €43 per participant² 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 [5] 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 Appendix A.3.3 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 [116] 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

²We 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: 1 IT Security consultant: 1	IT Security researcher: 1 Software security architect: 1

Table 3.2.: Quantitative Analysis: Demographics for $n=50$

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, 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 3.2. 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, $\mu = 8.9$, Md = 36, $\sigma = 8.9$), as well as a wide range of software development experience (min = 2 year, max = 30 years, $\mu = 12.96$ years, Md = 11 years, $\sigma = 8.21$). 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, $\sigma = 9.6$). More information on participants' demographics is available in the Appendix A.2.3.

3.2.3. 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 [46, 42, 47, 49]. 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" [47]. 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 [92]. After coding each transcription, the codes were updated and the guideline was adapted according to the results. We used theoretical sampling [57] 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*) [65, 66, 153, 89]. 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 [126, 141, 127, 46]. 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 [93]. 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” [46] 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” [46]. As suggested by Charmaz [46], we used the idea of theoretical sorting to develop a diagram out of our memos, themes and categories offering an initial analytic frame. 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” [46].
-

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 Appendix A.1.

In order to obtain the inter-coder agreement, we followed the approach of [44]. The coding schemes were compared, and inter-coder agreement was calculated using Cohen's kappa coefficient (κ) [53]. The agreement measured 0.81 (a value above 0.75 is considered to be a high level of coding agreement [87]).

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*.

3.3. 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.

3.4. 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 tempt 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

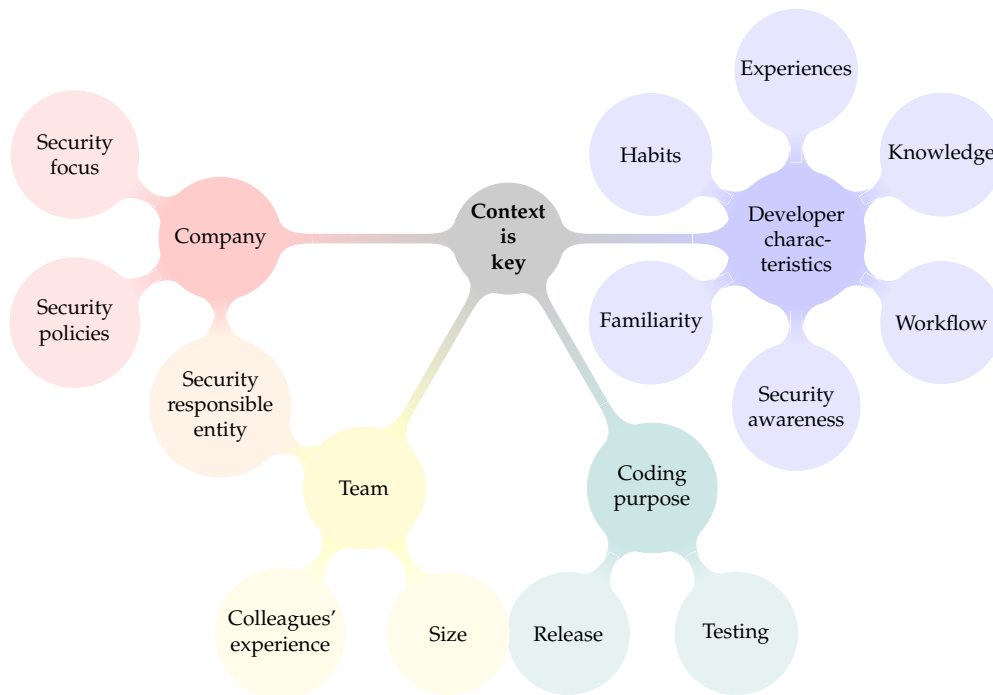


Figure 3.5.: Context aspects to consider for security warning design.

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., [25], 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 work 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.

3.5. 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 [56], 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 3.5 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.

3.5.1. 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. [189], 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.

3.5.2. 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 [107] 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.

3.5.3. 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

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

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. [130]. 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.

3.5.4. 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 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 a 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).

3.6. Quantitative Study

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

3.6.1. 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 Appendix A.3.

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 [84] 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 (S₁ and S₆) or a negative (S₂ and S₃) 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: $((S_1 + (7-S_2) + (7-S_3) + S_6) - 2)$.

3.6.2. 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

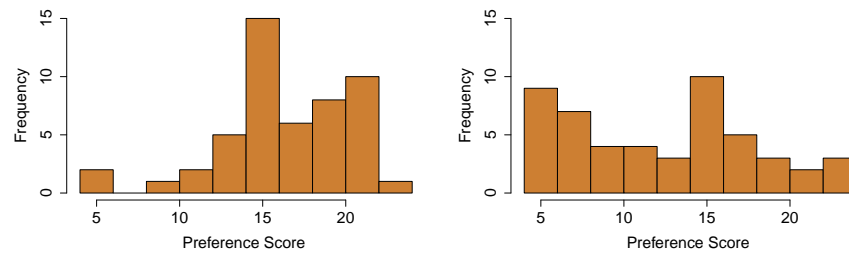
Looking at the mean values of the warning preference scores for our warning types, we can provide a ranking. The minimum score was 0 and the maximum score was 24.

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

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 ($p < 0.000^*$, $\chi_f^2 = 481.49$). 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 3.6a and 3.6b display the frequencies for the preference scores for warning markers and pop-up warnings. Figure 3.6a 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. 3.6b) 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 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 3.7. Here we plotted the



(a) Warning marker: warning preference score (b) Pop-up warning: warning preference score

Figure 3.6.: Histograms of warning preference scores (0-24)

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 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.

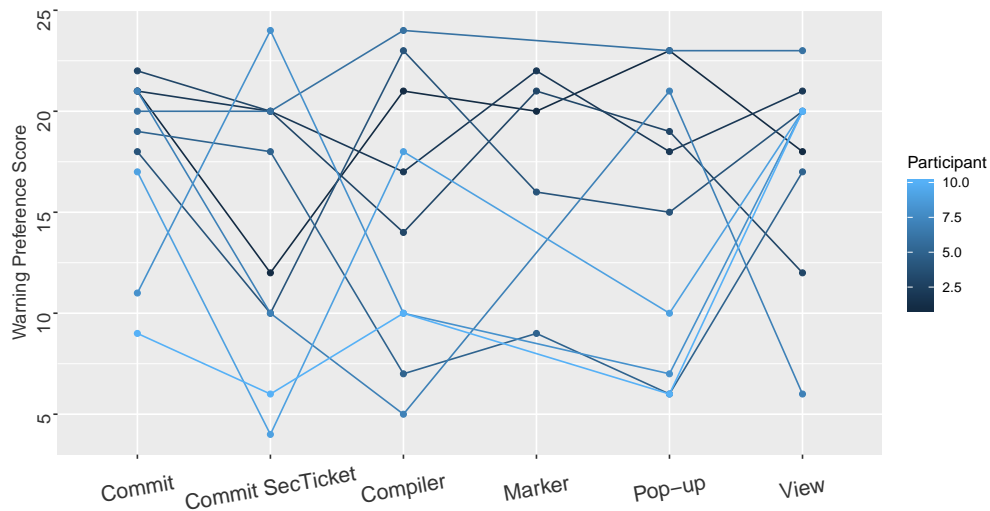


Figure 3.7.: Preference scores for 10 random participants

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 3.3.: Time preference of warnings

When to Display Warnings

Table 3.3 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 3.8 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.

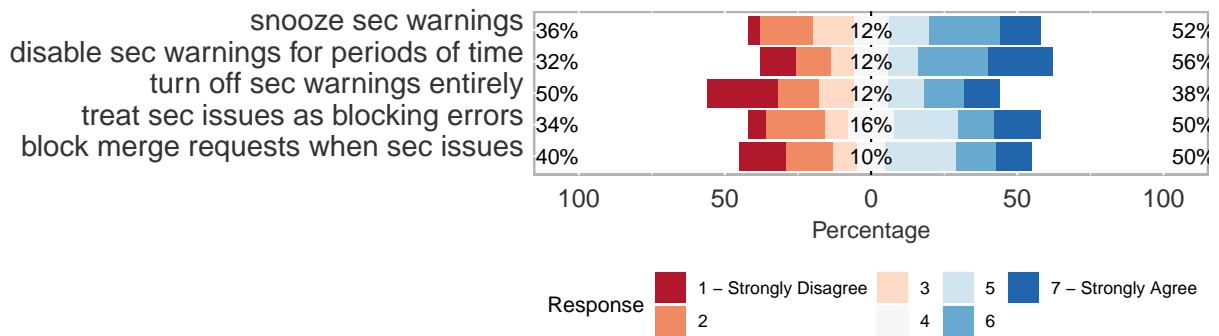


Figure 3.8.: Handling security warnings

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 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 [107] qualitative finding that workflow concerns matter, we found that the statement that a warning type suits a developers workflow (S_6 , Section 3.6.1) has a strong positive correlation with participants stating that they would like to be warned with this type of warning (S_1 , Section 3.6.1) ($p < 0.000^*$, $CI = [0.73, 0.81]$) with $r = 0.77$. This emphasizes that workflow compatibility is of utmost 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.

3.7. 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 [107] 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 [190] 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 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 [107] 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.

3.8. Summary

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.

4. Data Quality: Screening Questions for Online Surveys with Programmers

Disclaimer: The contents of this chapter were published as parts of the publication "Do you really code? Designing and Evaluating Screening Questions for Online Surveys with Programmers", presented at the 43rd International Conference on Software Engineering (ICSE'21) [60]. I led the conception of the study while Alena Naiakshina and Matthew Smith took part in the conception. Stefan Horstmann designed the questions under my supervision. I and Matthew Smith designed the study. Together with Stefan Horstmann, I conducted the study. The participants' survey answers were analyzed by me and Stefan Horstmann. I completed the statistical analysis of the data. Finally, I prepared the paper for publication with the cooperation from Alena Naiakshina, Stefan Horstmann, and Matthew Smith.

4.1. Motivation

Conducting user studies is an essential part of empirical software engineering; however, recruiting enough participants with programming skill can be challenging. Researchers use different methods to recruit programmers for studies related to software engineering. One common method is to recruit computer science (CS) students or developers from local companies for a local and in-person study. On the other end of the spectrum, researchers also commonly recruit developers on an international level for remote studies, using a variety of platforms. One advantage of recruiting locally is that researchers can be fairly certain that most of their sample actually has a computer science/programming background (e.g., [130, 131, 11, 140, 187, 50]). Unfortunately, finding enough willing participants locally can be difficult; Naiakshina et al. [131] reported that they could only get 40 out of 1600 CS students to take part in a study where the compensation was 100 euros. To widen the recruitment pool and include non-student participants, it is common for researchers to resort to online studies and recruit participants online (e.g., [186, 163, 94, 14, 132, 32, 25, 193, 133]). Diverse recruitment strategies have been used, such as cold-calling programmers on platforms such as Stack Overflow, GitHub, Meet-up groups, etc. or posting open invitations on social media, in forums, newsletters and events, with the expectation being that participants without programming knowledge will not sign up for the studies [25, 34, 164]. However, since researchers often offer significantly higher compensation than for end-user studies [130, 131, 132], there can be an incentive for participants to take part in a study despite having no programming skill.

The acquisition of skill is divided in three overlapping phases: (1) knowledge acquisition, (2) knowledge association, and (3) autonomous task performance [86, 35]. For *programming skill*, we use the definition of Bergersen et al. [35], which is in accordance to the definition used in psychology [86, 160, 20, 22, 21]: "the ability to use one's knowledge effectively and readily in execution or performance of programming tasks." In previous work with programmers, participants were often expected to have programming skill in various programming languages such as Python, Java, C, Perl, Haskell, JavaScript, PHP, etc. (e.g., [11, 186, 94, 14, 133, 35, 13, 28, 142, 129]). However, people who use programs such as Excel, manage a content management system (CMS) or write HTML might consider themselves as having programming skill. While this does not match our understanding and the requirements of previous research with programmers, we want to make it clear that participants stating that they have programming skill might be an honest misunderstanding

from our perspective and not necessarily malicious intent. None the less, having these kinds of participants in programming related studies and surveys is detrimental.

In studies which contain actual programming tasks, these participants can be detected fairly easily. However, since it is common practice to pay participants independently of how well they perform, they still cost time and cause financial damage to the researchers. More critically, studies which aim to examine attitudes towards software engineering related topics, such as new features of programming languages, API design, error handling, or security and privacy, might corrupt their data by including answers from participants who do not understand the subject matter because they do not have any programming skill. Examples for such studies are [25, 193, 34, 164, 179, 59, 188, 159, 129, 102, 138, 137, 28].

Researchers can also use online recruitment platforms such as Clickworker [1] or Qualtrics [7] for developer studies (e.g., [25, 59]). These platforms provide panels to recruit participants with specific skills, such as programming. However, as mentioned above, there might be misunderstandings concerning the term and the compensation for participants with (self-stated) programming skills is higher than for other types of study participants, and consequently there is the risk of participants falsely stating that they have these skills. We [59] came across such a case. We ran a survey to study developers' attitudes to security warning design, for which they used Qualtrics to recruit participants with programming skill. To test the participants' basic developer knowledge, we presented participants with some pseudocode that printed "hello world" backwards and asked what the output would be. Out of the 129 participants recruited on Qualtrics who stated that they had programming skill, only 33 gave the correct answer. Since most of the rest of the study consisted of closed questions, without the programming skill question, we would have included many participants in their data, who were not able to understand a very simple program.

To enable researchers to be more confident about conducting online surveys with programmers, we investigated the research question: *Which questions can be used to screen out participants without any programming skill (while meeting our requirements of effectiveness, efficiency, and robustness against cheating)?*

We created a survey instrument consisting of questions which can be used to assess whether a participant actually has programming skill. The questions are designed to take as little time as possible so they can be used in free or cheap pre-screening surveys. They needed to be so easy as to not annoy or challenge people with programming skill so we do not falsely reject participants, but also hard enough so that people without programming skill cannot make an educated guess or google the answer quickly.

We designed 16 questions of different types and tested them with different groups of participants: CS students, professional programmers, students enrolled in a behavioral economics study platform, as well as participants from Clickworker with and without self-stated programming skill. We evaluated the results from these groups to provide a shortlist of questions which seemed promising as screening questions for developer studies. To evaluate these questions, we tested them in an adversarial environment, where we offered non-programmers an extra monetary incentive to answer them correctly by any means, including using the Internet. As a final result, we offer a list of questions we believe can be used to screen non-programmers out of surveys with only minimal overhead for the participants with actual programming skill.

Table 4.1.: Overview of all questions

No	Question	Abbreviation	Category
Q1	Which of these lesser-known programming languages have you worked with before?	Unknown.Languages	Programming language recognition
Q2	Which of these websites do you most frequently use as aid when programming?	Source.Usage	Information Sources
Q3	Choose the answer that best fits the description of a compiler's function.	Compiler	Basic Knowledge
Q4	Choose the answer that best fits the definition of a recursive function.	Recursive	Basic Knowledge
Q5	Choose the answer that best fits the description of an algorithm.	Algorithm	Basic Knowledge
Q6	Which of these values would be the most fitting for a Boolean?	Boolean	Basic Knowledge
Q7	Please pick all powers of 2.	Power.of.2	Basic Knowledge
Q8	Please translate the following binary number into a decimal number 101.	Bin.Conv	Number Formats
Q9	Please select all even binary numbers.	Bin.Even	Number Formats
Q10	Please select all valid hexadecimal numbers.	Hexa.Num	Number Formats
Q11	When multiplying two large numbers, your program unexpectedly returns a negative number. What might have caused this?	Error.Overflow	Finding Errors
Q12	What is the run time of the following code?	Runtime	Algorithmic runtime
Q13	When running the code, you get an error message for line 6: Array index out of range. What would you change to fix the problem?	Error.OutOfBounds	Finding Errors
Q14	What is the purpose of the algorithm?	Sorting.Array	Code Comprehension
Q15	What is the parameter of the function?	Function.Param	Basic Knowledge
Q16	Please select the returned value of the pseudo code.	Backward.Loop	Code Comprehension

4.2. Methodology

In this section, we present the questions we designed to identify participants with programming skill and the studies we ran to evaluate it.

4.2.1. Instrument Requirements

Unlike Bergersen et al.'s [35] instrument to determine programming skill which takes two days to complete, our goal is to assess whether participants have programming skill or not with as little effort as possible. For the questions to be used in pre-screening surveys or as quality control questions, we must ensure that they take people with programming skill only a few minutes at most to answer. Therefore, our requirements regarding the instrument were as follows:

- **Effectiveness:** The instrument should be able to differentiate between programmers and non-programmers. Hence, the questions need to rely on domain knowledge and be complex enough so that only programmers can answer in a reasonable amount of time. It should not leave any scope for mere guesses.
- **Efficiency:** The instrument should consume as little time as possible. So, the goal is to frame questions that programmers can answer quickly. It is also desirable if it would help the participants without programming skill to decide quickly that they cannot answer the question, since we do not want to waste their time either.
- **Robustness against cheating:** The instrument should be designed in a way that it becomes difficult for participants without programming skill to come by the answers, for instance, by using online search engines or forums on which Clickworkers exchange information about studies.
- **Language independence:** The instrument should work regardless of the programming language the participants are skilled in. While it might also be useful to filter based on specific languages, that is beyond the scope of this work.

4.2.2. Survey

We used Dillman's pre-testing process to develop an online survey with 16 different questions [67, p.140-147]. The Dillman's pre-testing process is a three-step approach to design survey questions

in general. It includes literature review, using a think-aloud approach and conducting a pilot study. First, we reviewed related work and decided on different question types. We started with a larger pool of questions from related work. Since these questions rarely met all our requirements (e.g., time and effort), three researchers from Computer Science in different positions (graduate student, PhD student, professor) developed further questions by considering related work, but also examining the main concepts of programming. Second, another researcher went through the questions using a think-aloud approach. Third, we conducted a pilot study with two participants. A summary of our questions can be found in Table 4.1. The full questions can be found in the Appendix B.2. A total of 16 questions were created and put under a number of categories:

- **Programming language recognition (Q1):** Q1 is a two part question. The first part asks the participants to self report their programming language skill for “well-known programming languages” such as C, C++, or Java. This part does not need to be evaluated for the instrument. The second part asks the participants to self-report their programming language skill for “lesser-known programming languages” such as Torg or Yod. However, all but one of the lesser known programming languages are fake. Both the parts offer a “none of the above” option. The idea behind this question is to see how quickly people with programming skill select their languages from the first list and then realize that they do not know any from the second. Our hypothesis is that they will probably also realize that most of the names on the second list are fake and select “none of the above,” while non-programmers who want to falsely claim skill would select a couple.
- **Information sources (Q2):** Previous studies have analyzed what kind of websites developers use while programming, with the most popular being Stack Overflow [85]. Hence, Q2 contains Stack Overflow and some decoy options. Participants with programming skill can quickly pick Stack Overflow, while non-programmers might not be aware at all.
- **Basic knowledge (Q3-Q7, Q15):** These questions cover general programming knowledge, for instance, regarding what a compiler does, what an algorithm is, and what a recursive function is.
- **Number formats (Q8-Q10):** We asked simple questions related to hexadecimal and binary conversion. Most computer science programs or programming language tutorials deal with binary and hexadecimal numbers; hence, most non-programmers will not have much experience with this.
- **Finding errors (Q11, Q13):** A good test of programming skill would have been asking the participants to actually write some code; however, that would cost more time than can be permitted and is hard to automatically verify. Therefore, as an alternative, we asked the participants to find errors in code snippets or explain why the errors occurred.
- **Algorithmic runtime (Q12):** We also added a basic question about the run time of some simple pseudocode.
- **Program-comprehension (Q14, Q16):** We showed the participants two pseudo-algorithms and asked them about their functionality and output. It is important to note that Q16 was taken from our prior study [59].

The questions Q13+Q14 as well as Q15+Q16 are based on the same pseudocode. Hereafter, we will refer to each pair of these questions as a “question block.” While we tried to keep the time

spent on each question short, especially for participants with programming skill, we also had to enable automatic evaluation and make the questions robust against random guesses. Thus, we opted for closed multiple choice questions with 5 to 6 possible answers for each question. Most questions had one unambiguous correct answer. Exceptions were 2 of the number-format questions, where participants were asked to select all correct solutions. The incorrect answers were chosen in a way as to look plausible to participants without programming skill.

Furthermore, an attention check question [116] appeared randomly during the survey to filter out careless respondents. For the attention check question—This is an attention check question. Please select the answer “Octal”—the correct item needed to be picked. The questions and answer options were shown in a randomized order to mitigate response fatigue and response order effects [118].

We used 2 versions of this survey. The initial version included “I don’t know” or “I don’t program” answer options. We included these options because we wanted to minimize guessing at this stage so we could get an accurate view of what non-programmers state did not know. The second version of the survey was conducted in an adversarial setting (see Section 4.6.2). Here, the participants were given a monetary reward for each correct answer and the “I don’t know” or “I don’t program” options were removed. This setting simulated a screening setting in which non-programmers might try to guess the correct answers to take part in a well-compensated survey.

After completing the programming questions, the participants were asked to answer demographic questions, including ones related to their age, job, and programming experience. The full questionnaire can be found in the Appendix B.2. For evaluation and for testing our time requirement, we set a timer for each question to measure how much time was spent to solve it.

4.2.3. Statistical Testing

We categorized the answers as *correct* or *incorrect*. In order to test whether the different groups had different success rates for different questions, we used the Fisher’s exact test (FET) [80, p. 816] on each question. We reported confidence intervals (CI) and odds ratio (OR) to interpret the details of the tests. We corrected all Fisher’s exact tests for multiple testing using the Bonferroni-Holm correction.

To analyze the entire set of questions, we used latent class analysis, as it is suited the categorical data [123, 54, 119]. The latent class analysis reveals whether the data shows a number of distinct classes. Our assumption was that we will get two: participants with and without programming skill. We tested the models with more classes as well but selected the one with the lowest Bayesian information criterion (BIC).

4.2.4. Participants

Recruitment

We sampled through different channels to obtain data from different groups. We sampled 17 CS students using the mailing-list of an advanced programming lecture from the undergraduate program of our university. As compensation, the students received bonus points for their exam admission. All the CS students passed our attention check.

Table 4.2.: Demographics of the participants (n = 249)

Group	Sample	n	Gender	Age	Country of Residence	General Programming [years]	Programming Experience
Programmer	<i>CS students</i>	17	Female: 4, male: 13	min: 19, max: 30, mean: 21.82, md: 20, sd: 3.26	Germany: 17	min: 2, max: 16, mean: 5.31, md: 4.5, sd: 3.41, NA: 1	
	<i>Professional developers</i>	33	Female: 2, male: 31	min: 25, max: 55, mean: 36.45, md: 36, sd: 8.04	Germany: 31, Austria: 2	min: 2, max: 30, mean: 13.09, md: 15, sd: 7.31	
Non-Programmer	<i>Econ students</i>	50	Female: 36, male: 13, PNTA: 1	min: 18, max: 28, mean: 22.66, md: 23, sd: 2.3	Germany: 49, NA: 1	min: 0, max: 2, mean: 0.21, md: 0, sd: 0.52	
	<i>Clickworkers without programming skill</i>	50	Female: 20, male: 29, PNTA: 1	min: 19, max: 67, mean: 34.02, md: 31.5, sd: 10.43	Germany: 21, UK: 8, USA: 6, Other: 15	min: 0, max: 25, mean: 1.63, md: 0.25, sd: 4.1	
Test Group	<i>Clickworkers with programming skill</i>	52	Female: 10, male: 41, PNTA: 1	min: 18, max: 58, mean: 33.73, md: 31.5, sd: 9.81	Germany: 22, UK: 5, Other: 25	min: 0, max: 30, mean: 6.08, md: 3, sd: 7.78, NA: 1	
Attack Scenario	<i>Clickworkers without programming skill</i>	47	Female: 18, male: 28, PNTA: 1	min: 19, max: 54, mean: 32.15, md: 30, sd: 9.15	Germany: 16, Italy: 4, Spain: 4, USA: 4, Other: 19	min: 0, max: 26, mean: 2.22, md: 1, sd: 4.82	

md: median. PNTA: prefer not to answer. See Appendix B.3 for further details on occupation and country of residence.

Additionally, we invited 49 professional developers from personal contacts and from a database of professional developers who took part in our past programming studies and agreed to take part in future studies. Thirty-five participants completed the survey. We excluded one from our data set because we were not able to identify the participant on our invitation list, and it seemed like that the study link was forwarded. All the professionals passed our attention check and received 10 euros for their participation. We combined these two groups to form our ground truth since we knew for sure that they all have programming skill.

Next, we recruited 54 students in cooperation with the behavioral economics group from our university. They have a recruitment system which sends email invitations for studies to enrolled users. The majority of them were economics students; however, others potentially including computer science students could have enrolled as well. We refer to these participants as *econ students*. Of these 54 participants, 50 passed the attention check. Based on the question of self-reported programming experience, 10 participants had at least some (0.5 years) experience. The participants received 5 euros as compensation.

We also recruited 75 participants from Clickworker, who did not have any programming skill. Of these 75 participants, 53 passed the attention check and 50 completed the survey. They received 2.50 euros for their participation.¹ However, in contrast to all other samples, we used the default Clickworker invitation description which cautions Clickworker participants that attention check questions needed to be solved correctly to receive the payment. We accepted this difference since this is the norm on Clickworker; it is the norm to pay participants even if they fail the attention check questions on the other recruiting platforms.

¹This fulfills the minimum wage requirement. Our compensation was higher than the recommendation of the platform, which was 1.50 euros for 10 minutes.

We combined the last two groups as our ground for participants without programming skill. However, the situation is not as clear cut as with the programmers above since CS students can also be enrolled in the econ platform, and both econ students and regular Clickworker participants might have programming skill even though they do not state it. In this combined group we have 35 out of 100 participants who stated that they have programming experience. However, since we have no way of verifying this properly, we did not remove them from our evaluation, to be on the conservative side.

Further, we recruited 55 Clickworker participants who listed programming as a skill in their profile. The hiring conditions were the same as above. Of these 55 participants, 52 passed the attention check. While the above-mentioned groups were used to design our screening instrument, we used this last group as a real-life test to see how many of these would pass our screening questions.

Finally, we tested the 8 best questions using an attack scenario with 51 participants from Clickworker, of whom 47 passed the attention check. We paid a base compensation of 2 euros to Clickworkers who did not state that they had programming skill and provided a bonus payment of 2 euros for each of the 8 correct answers. This should simulate a non-programmer adversary who wants to pass a screener question to be able to take part in a well-compensated developer study.

Demographics

The demographics of our tested groups can be found in Table 4.2. Of all the 249 participants, 155 were male, 90 were female, and 4 preferred not to answer the question. From the programmer group and the test group, almost all participants were male (developer: 44/50 male; test group: 41/52 male). By contrast, from the non-programmer group, more participants were female (out of 100: 56 female, 42 male, and 2 preferred not to tell). The majority of the participants were from Germany.

While professional developers reported to have on average 13.09 years of programming experience (min: 2, max: 30, median (md): 15, standard deviation (sd): 7.3), CS students indicated, on average, 5.31 years of experience (min: 2, max: 16, md: 4.5, sd: 3.41). All the participants from the programmer group indicated to have worked with Java before. A total of 31 of the 50 programmer participants indicated to have worked with JavaScript, 34 with C, 30 with Python, 28 with C++, 22 with PHP, 20 with C#, 19 with Shell, 8 with Typescript, 3 with Ruby, another 3 with Groovy, and 2 with Go.

Clickworker participants who claimed to have programming skill in their profile indicated in our survey to have, on average, 6.08 years of experience (min: 0, max: 30, md: 3, sd: 7.78). By contrast, Clickworker participants who did not indicate to have programming skill in their profile reported in the survey to have, on average, 1.63 years of experience (min: 0, max: 25, md: 0.25, sd: 4.10). Finally, most of the econ students reported not to have programming experience at all (mean: 0.21, min: 0, max: 2, sd: 0.52).

4.3. Ethics

The institutional review board of our university approved our project. The participants of our study were provided with a consent form outlining the scope of the study and the data use and

retention policies, and we also complied with the General Data Protection Regulation (GDPR). The participants were informed of the practices used to process and store their data, and that they could withdraw their data during or after the study without any consequences. The participants were asked to download the consent form for their own use and information.

4.4. Limitations

We compensated each sample differently, as the different groups had different payment expectations. However, it could be that the different compensation levels affected the results. We found a couple of participants in the non-programmer group who looked like they had programming skill. There might also have been some with programming skill whom we did not recognize. However, we think that the number of programmers that accidentally fell in the non-programmer group was not significant enough to interfere with our study. If anything, our instrument's performance will be under-reported since we count any unknown programmer in the non-programmer group who is identified as a programmer as a failure for our instrument. While we have recruited a mix of CS students, econ students, professional developers, and Clickworkers with and without programming skill, we do not claim that this is representative for all programmers. Hence, further studies will be needed to extend and validate our results.

4.5. Results

In this section, we describe the effectiveness and efficiency of our 16 defined screener questions. However, the questions should also be effective in a way that the number of programmers who fail and the number of non-programmers who either know or guess correctly should be low. Therefore, we additionally tested our questions with a test group to reduce our question set to the most promising questions and evaluated them in an adversarial scenario.

4.5.1. Effectiveness

Except for Q1, all Fisher's exact tests (15/16) were highly significant even after the Bonferroni-Holm correction. This, in turn, indicates that the remaining 15 questions were different in the distributions of correct and incorrect answers between the non-programmer and programmer groups.

We conducted a latent class analysis on the 16 questions. We chose a model with two groups ($G^2(2)$: 613.02 (Likelihood ratio/deviance statistic) $\chi^2(2)$: 174111.9 (Chi-square goodness of fit), maximum log-likelihood: -907.89, entropy: 6.06), since the BIC was lower for models with more classes and it fit well to our assumption. Figure 4.1 visualizes the proportion of probabilities for choosing the correct or incorrect items according to the groups. The predicted class shares were 0.62 for Class 1 (non-programmer) and 0.38 for Class 2 (programmer). Thus, according to the class analysis, two classes can be distinguished within this sample. This fits into our self-chosen samples, since we had 50 programmers and 100 non-programmers sampled. Indeed, our questions are applicable to split a population according to the answers the participants provided. In the following, we describe the effectiveness of each question in detail.

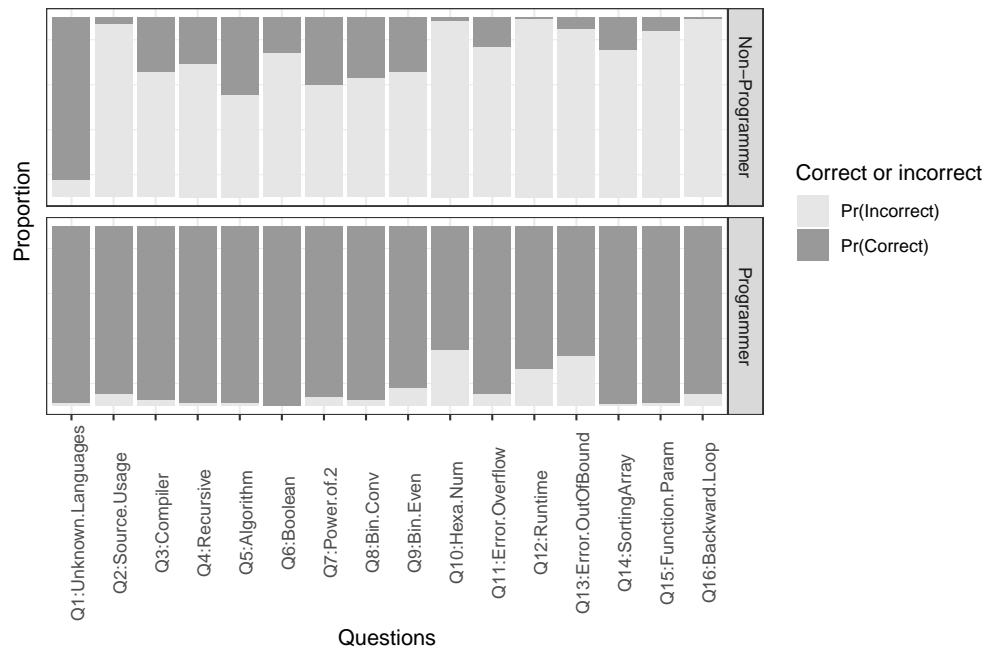


Figure 4.1.: Plot of latent class analysis with the programmer and non-programmer groups.

Programming Language Recognition (Q1)

We found that almost all the non-programmers (91% or 91 of 100) and programmers (98% or 49 of 50) answered the question for lesser-known languages correctly. That means in this case, they chose the answer “None of the above.” Nine participants from the non-programmer group and one participant from the programmer group selected non-existent programming languages. This shows that without incentive most non-programming participants had no reason to pick fake languages, although it is interesting that roughly 10% of non-programmers selected some of the fake languages.

Information Sources (Q2)

All the 50 programmers (100%) selected Stack Overflow as one of the most used aids for programming. No other source was reported. Most participants from the non-programming group reported that they do not program (60% or 60/100), while 15 said that they do not use any of the listed websites in the questionnaire for programming. Interestingly, 9 participants chose MemoryAlpha, 8 selected Wikipedia, 2 other participants marked LinkedIn and only 6 participants from the non-programmer group picked Stack Overflow as their answers.

Basic Knowledge (Q3 to Q7 and Q15)

First, we analyzed the answers given on the description of a compiler (Q3). All the participants from the programmer group and 33% participants from the non-programmer group chose the correct

answer. As stated earlier, our non-programmer group is not as controlled as our programmer group because 35 participants from the group stated that they had some programming skill. However, of the 33 correct answers only 23 came from this group, i.e., 12 participants who stated to have programming skill did not give the right answer for Q3, whereas 10 who stated that they had no skill got it right. This similar pattern that emerged was found in all the knowledge questions with no clear distinction visible between the two subgroups. Thus, for simplicity we will not report further on this, but a full overview can be found in the Appendix B.2.

Second, we analyzed the answers given on the definition of a recursive function (Q4). The answers were very similar to the compiler question. All programmers could answer the question correctly, whereas 30% of the non-programmers got the correct answer. The performance for Q5 that was about the definition of an algorithm (Q5) was even worse, as 47 (out of 100) were able to answer it correctly. One programmer failed the question. The effect on the boolean question (Q6) was significant as well, since all the programmers successfully chose the correct answer, while only 25% of the non-programmers answered correctly.

We also included a question (Q7) about the power of two, since we thought most programmers work with powers of two. The answers were only marked as correct if all the powers of two were selected. Two programmers answered incorrectly, while the rest chose the correct answer. From the non-programmer group, 41% managed to pick the right powers of two as well. Furthermore, we asked our participants to pick what a function's parameter is (Q15). This question was answered correctly by all the programmers, while only 13 participants from the non-programmer group selected the correct answer. It seems that programmers were familiar with the definition, while the non-programmers struggled to answer this question correctly.

Number Formats (Q8 to Q10)

First, we asked our participants to convert a number from the binary system into the decimal system (Q8). Forty-eight of the 50 programmers and 38 of 100 non-programmers solved the question correctly. With regard to picking all the even binary numbers question (Q9) with multiple answers, 5 programmers failed to select all the correct answers. However, 34 out of 100 participants from the non-programmer group got the correct answer. To test the participants' knowledge with hexadecimal numbers, we asked them to select all the valid hexadecimal numbers in Q10. To get the correct answer, multiple choices needed to be selected. While the effect of groups on the answers was again significant, this question seemed to be very challenging for all the participants. 30% (15 of 50) of the programmers were unable to solve this question. From the non-programmer group, only 6 participants succeeded in selecting the correct answers.

Finding Errors (Q11 and Q13)

Fixing bugs takes a large percentage of a programmer's working time. With regard to the question of what could happen if two large numbers are multiplied and a negative number is returned (Q11), the difference of the two groups was significant. Forty-seven of the 50 programmers were able to answer this question correctly as well as 21 of 100 non-programmers. We also investigated the common errors that programmers face during programming and requested them to solve an

ErrorOutOfBounds (Q13). Only a few non-programmers answered the question correctly (9 of 100). However, the programmers also seemed to have trouble with this question, because “only” 38 out of 50 selected the correct answer.

Algorithmic Runtime (Q12)

The question for the run-time seemed to be very difficult for both the groups. We concluded that even the participants with programming skill were not very familiar with algorithmic run-times. The effect of the group variable was significant because the proportion of correct answers differed between both the groups.

Program-comprehension (Q14 and Q16)

Comprehension of the sorting array pseudocode seemed to be an easier task for the programmers, because almost all of them answered Q14 correctly (49 of 50). From the non-programmer group, 24 selected the correct answer. Furthermore, 3 programmers were unable to solve the “hello world” pseudocode task (Q16) correctly. Additionally, only 7% of non-programmers choose the correct answer. Consequently, the difference in correct answers between both the groups was significant.

4.5.2. Efficiency

The participants in the non-programmer group recorded 7.87 minutes as the median time to complete the whole questionnaire, while the participants in the programmer group finished the survey with a median time of 10.87 minutes. All questions showed a mean under 100 seconds; thus, all questions fulfilled our efficiency requirement. We found that answering knowledge questions took the least time, while answering questions about number conversion took longer. As expected, each of the two question blocks including pseudocode (Q13+Q14 and Q15+Q16) took more time as compared to other multiple choice questions. We found that participants with programming skill needed more time to answer them. The reasons for this could be that non-programmers selected “I don’t know” or gave a random answer quickly, since they knew they could not understand the code. The adversarial measurements in Section 4.6.2 are more relevant for these questions.

A visualization of the mean times for both programmer and non-programmer groups according to each task block, an overview for the number of correct and incorrect answers of the programmer and non-programmer groups for each question as well as the statistical analysis summary are available in the Appendix B.1, B.3.

4.6. Testing the Instrument

We tested the instrument in two scenarios: (1) Non-Adversarial Test group and (2) Adversarial Attack group. First, we tested our survey instrument with a set of 52 participants from Clickworker who indicated in their profiles to have programming skill. This scenario is close to how real studies would be conducted, i.e., researchers use a platform like Clickworker to recruit participants who state that they have programming skill. We did not offer significantly higher compensation to minimize the incentive to claim skill to participate in our survey. We also included in the “I don’t know” options, since we wanted to see how many of the self-reported programmers from

Clickworker would choose that they did not know answers in a non-adversarial setting. The results also served for comparison of self-reported programming skill of the participants from Clickworker with professional developers from our controlled sample. Later, we selected the most promising questions and tested them in an adversarial scenario, where we recruited 47 participants from Clickworker who did not state to have programming skill. In the introduction, we explained the goal of our study and asked the participants to try and pass themselves off as programmers. To incentivize them, we paid a base fee of 2 euro and an additional 2 euro for every correct answer for the 8 questions. In this scenario, we removed the “I don’t know” options because we needed to evaluate the questions in an adversarial setting and measure their guessability. This scenario simulates people without any programming skill trying to break our screener questions to take part in a well compensated developer study.

4.6.1. Non-adversarial Test Group

Programming language skill (Q1): Forty-eight of the 52 participants did not select any imaginary languages when asked for skill with lesser-known programming languages. The majority (46) selected “None of the above” and 2 selected “SHROUD.”

Information sources (Q2): For answering the question for the most used information sources, 30 of the 52 participants selected Stack Overflow, while the rest chose that they did not program or have not used any of the websites suggested (9 of 52). Five chose Wikipedia.

Basic knowledge (Q3 to Q7 and Q15): Forty-one of the 52 participants correctly selected the description of a compiler (Q3), whereas 11 failed. For the description of a recursive function (Q4) and the value of a boolean (Q6), 41 selected the correct answer. For the description of an algorithm (Q5), only 5 failed to select the correct answer. Forty participants were able to select all the powers of two (Q7) while 10 did not. Thirty-three participants selected the function’s parameter (Q15) correctly.

Number Formats (Q8 to Q10): Forty participants were able to select the correct answer for a simple binary conversion (Q8), while 6 selected a wrong answer and another 6 reported to not know the answer. The multiple response questions seemed to be more difficult. Thirty-two participants could correctly select all the even binary numbers (Q9). Further, only 30 participants selected all the valid hexadecimal numbers (Q10), while 17 failed or selected “I don’t know” (5).

Error (Q11 and Q13): Thirty-five participants correctly selected an overflow as source of error in Q11, and 12 participants reported that they did not know the answer and the rest selected an underflow. For the array out of bound error (Q13), only 17 of 52 (32%) participants were able to select the correct solution.

Algorithmic runtime (Q12): The runtime question seemed to be the hardest one for the participants from the test set. Only 15 answered correctly, 21 did not know the answer and the rest selected a wrong response.

Program-comprehension (Q14 and Q16): Thirty-one participants selected the correct purpose of our sorting array pseudocode (Q14). Interestingly, only 24 participants selected the correct output of the hello world pseudocode (Q16), whereas 13 participants selected the wrong answer “hello world.”

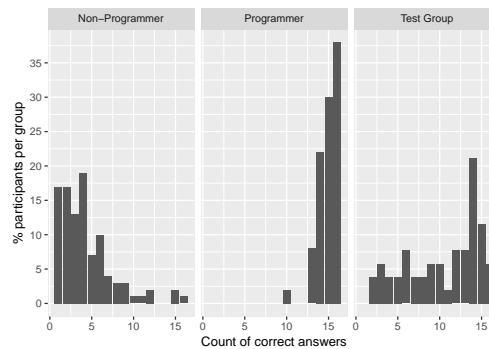


Figure 4.2.: Number of correct solutions of all 16 questions per group.

Comparison of the Groups

Table 4.3 shows the correct answer rates of the programmer, non-programmer, and the non-adversarial test group for the 16 questions. The test group (Clickworker who stated that they had programming skill) always had more correct answers than the non-programmer group (econ students and non-programmer Clickworkers) but less than the programmer group (CS students and company developers). This suggests that some Clickworker participants do not interpret programming skill the way we do or falsely indicated in their profile to have programming skill. In either case, it is important to be able to identify these participants during screening. Interestingly, 7 participants from this test group selected “*I don’t program,*” when we asked for their level of programming expertise at the end of the survey, despite listing programming as a skill in their profile. The number of correct solutions for all the 16 questions per group is visualized in Figure 4.2.

4.6.2. Attack Scenario

Based on the above findings, we selected the most promising questions (Q2 to Q4, Q6, Q14, and Q15) by excluding all the questions that did not achieve a correct answer rate of at least 98% from the programmer group. We also excluded all the questions where more than 40% of the non-programmer group gave the correct answer (Table 4.3). We chose these cut-offs for several reasons. First, we did not require 100% correctness from the programmers because even programmers make mistakes. Additionally, we allowed some false positives from the non-programmers because we need to keep the questions quick and simple to not lose programmers to the screening process. Since, in most scenarios, we would recommend the use of multiple questions, the false-positive rate will be lower due to the combination. In addition to the 6 questions mentioned above, we included Q1 and Q16 to our set of most promising questions because we expected them to perform better in the attack scenario. After each question, we asked the participants whether they looked up the answer on the Internet or solved it on their own (see Appendix B.4 for more details).

Programming language skill (Q1): In the adversarial setting, we tested Q1 in two versions: 1) both parts of the questions (real and fake programming languages) and 2) on its own (only fake programming languages). Both versions were ineffective with only 4 of 18 in version 1 and 3 of 29

Table 4.3.: Percentages of correct answers in each group.

No	Question	Programmers	Non-Programmers	Test Group
Q2	Source.Usage	100%	6%	57.69%
Q15	Function.Param	100%	13%	63.46%
Q6	Boolean	100%	25%	78.84%
Q4	Recursive	100%	30%	78.84%
Q3	Compiler	100%	33%	78.84%
Q14	Sorting.Array	98%	24%	59.61%
Q5	Algorithm	98%	47%	90.38%
Q1	Unknown.Languages	98%	91%	96.15%
Q8	Bin.Conv	96%	38%	76.92%
Q7	Power.of.2	96%	41%	76.92%
Q16	Backward.Loop	94%	7%	46.15%
Q11	Error.Overflow	94%	21%	67.30%
Q9	Bin.Even	90%	34%	61.53%
Q12	Runtime	80%	6%	28.74%
Q13	Error.OutOfBound	76%	9%	32.69%
Q10	Hexa.Num	70%	6%	57.69%

The table is sorted descended by the column "Programmers" and ascended by the column "Non-Programmers," because the most promising questions require to be correctly answered by programmers and incorrectly by non-programmers.

Table 4.4.: Overview of screening question recommendations for programming skill

Question	Recommended	Suggested time limit [seconds]	Excluded programmers with time limit (n = 50)	Attackers (n = 47) (included excluded)
Q1	✗	-	-	-
Q2	✓	30	2	10 19
Q3	✓	60	3	22 14
Q4	✓	30	4	8 28
Q6	✓	30	0	14 22
Q14	✗	-	-	-
Q15	✓	Not necessary	-	-
Q16	✓	Not necessary	-	-

The table shows an overview of our recommendations for the eight questions tested in the attack scenario. Colors: red = not recommended, green = recommended without restrictions, yellow = recommended but with time limit

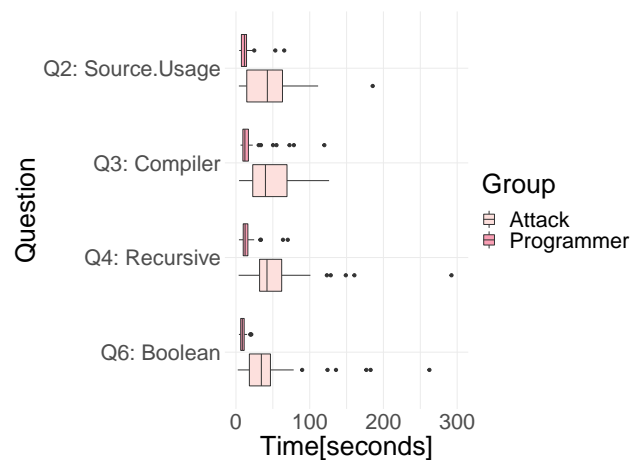


Figure 4.3.: Comparison knowledge questions: Time to solve each question **correctly**.

in version 2 where the participants chose non-existent programming languages. Our assumption that non-programmers would pick these turned out to be false.

Sources (Q2): Twenty-nine of the 47 Clickworkers (61%) reported "Stack Overflow" to be their most used information source for programming. Six chose "None of the above," while 7 chose "Wikipedia" and 5 "MemoryAlpha."

Compiler's function (Q3): Thirty-six of the 47 participants (76%) chose the correct answer for the functionality of a compiler.

Recursive function (Q4) and boolean (Q6): 76% (36 of 47) participants correctly defined a recursive function and answered the value that a boolean can take.

Sorting array (Q14): Twenty-nine of the 47 participants (62%) answered the sorting algorithm question correctly. This might be the first indication of algorithms being more difficult for the participants without programming skill to look up.

Parameter of a function (Q15): Thirteen of the 47 participants (27%) selected the correct answer.

Hello World (Q16): 25% of Clickworkers (12 of 47) picked the correct answer for the hello world question.

Our analysis showed that Q15 and Q16 performed best according to the correct answer rates in the attack scenario. Figure 4.3 shows the time taken to solve the questions correctly for the programmer and the attack groups. We excluded Q1 since we tested 2 versions of this question, as described above, and the question performed so poorly that it was not considered. We also did not directly compare Q14, Q15, and Q16 to the original questions since, in the prior analysis, they were part of a question block. In the attack scenario, we split them in order to get the information for each question whether participants googled the answers or not. All knowledge questions, except Q2 ($p = 0.054$), showed a significant time difference (wilcoxon rank sum test) between the developer and attacker groups, with the attackers taking significantly longer.

4.7. Discussion and Recommendations

When conducting our non-adversarial evaluation with the Clickworker test group, we only selected participants whose profiles included programming skill. The results of this test group demonstrated

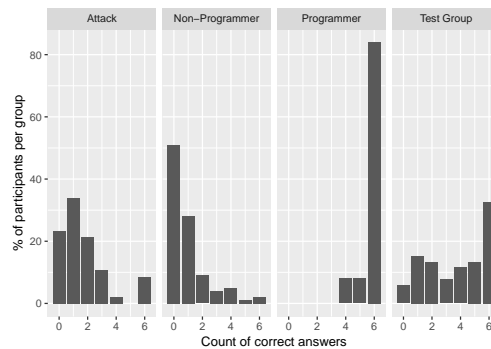


Figure 4.4.: Distribution of correct solutions of all 6 recommended questions per group with time limits applied as in Table 4.4.

that for the kind of developer studies that are common in our community, it is not recommendable to rely on the self-reported programming skill or a platform’s recruitment features. In our test set, 42% of the Clickworker programmers got fewer correct answers than the poorest performers in our ground truth programmers group. They mostly got all the answers right while many Clickworker developers got less than half right answers. Considering the small sample sizes, which are common in developer studies, even a small amount of noise can mask true effects or worse create false effects. Having potentially one-third participants without programming skill in a developer survey can cause significant disruption, and we highly recommend using screening questions to avoid this. The fact that we removed 10 of our 16 questions, since they proved to be less effective than we had hoped, suggests that the current practice of some researchers creating and using untested screener questions is sub-optimal. We hope that our tested questions can be a first step toward creating a common screener instrument for our community.

We recommend the use of screener questions for studies targeting people with programming skill, especially if these studies do not contain a programming task. Table 4.4 summarizes our screener question recommendations. The most effective but also the slowest questions were the code comprehension questions, i.e., Q15 and Q16. If time penalty is acceptable, we recommend using these or similar questions as screeners. If this is not feasible, we recommend to randomly use one of the four knowledge questions (Q2 to Q4 and Q6) with a time limit, since our attack group demonstrated that these question can be looked up. The time limit can be used to configure the false rejection participants with programming skill and the false acceptance of those without. We recommend a 30-seconds time limit for Q2, Q4, and Q6 and a 60-seconds time limit for Q3. In the Appendix B.3, B.4, B.5, B.6, we provide details of how we chose these time limits. Figure 4.4 shows the distribution over the 6 recommended questions of correct solutions in the groups with the time limits applied.

4.8. Summary

In previous online studies with programmers, researchers often relied on participants’ claims to have programming skill or they used programming tasks or programming knowledge questions to verify these. Our work showed, however, that designing programming screener questions is not

trivial and we would not recommend using questions without testing them before. While we raised a methodological problem in software-engineering work on a meta-level, we also contributed concrete and validated screener questions on a primary level.

We surveyed a total of 249 people to find questions that can be used to filter participants with programming skill. To get a ground truth sample of programmers, we selected participants for whom we were able to verify that they actually have any programming skill. We, then, recruited non-programmers and Clickworkers with and without self-reported programming skill to test our screening instrument. Finally, we tested our instrument under adversarial conditions to test its robustness. Based on our evaluation, we recommend 6 of our 16 screener questions for use in online studies.

In future work, we will continue to expand and test our question set. While the small set is sufficient to protect against non-adversarial participants, who simply have a different interpretation what programming is than we do, a larger set will be more robust in an adversarial setting.

5. Data Quality: Time Limits in Screener Questions with Programmers

Disclaimer: The contents of this chapter are going to be published as parts of the publication "Testing Time Limits in Screener Questions for Online Surveys with Programmers", which is going to be presented at the 44th International Conference on Software Engineering (ICSE 2022) [62]. I led the conception of the study while Alena Naiakshina and Matthew Smith took part in the conception. Stefan Horstmann designed the new questions under my supervision. The study was conducted by Stefan Horstmann. The participants' survey answers were analyzed by me and Stefan Horstmann. I led the statistical analysis of the data and Stefan Horstmann completed it. I prepared the paper for publication with the cooperation from Alena Naiakshina, Stefan Horstmann, and Matthew Smith.

5.1. Motivation

User studies can be an important tool to examine the issues of certain programs. However, recruiting participants for developer studies may prove difficult, especially if participants must possess programming skills for the studies. In some cases, computer science students are recruited for surveys and tests, yet researchers may also be required to recruit professionals [130, 131, 132, 133, 58]. Researchers have been recruiting programmers from known IT-firms [128, 124], local programming groups [11] or GitHub [186, 94, 14, 97, 13].

Some online services like Clickworker [52], Qualtrics [7] or Amazon MTurk [173] offer recruitment options for online surveys with the possibility to choose participants with programming skills. However, as the skill level is only self-reported, participants may misrepresent their programming skills to attain monetary compensation or by accident. Whereas these participants would be identified relatively quickly during practical tasks, online surveys may only include multiple-choice questions, and participants without programming skills can threaten the validity of these studies. However, these participants might be filtered out during the survey process, using tasks that are easy to solve by programmers but difficult for non-programmers. These tasks should make sense in an online survey and require as little time as possible to be solved by programmers.

Sixteen multiple-choice questions have already been developed and tested in [60]. They tested the tasks with programmers and non-programmers. They also simulated an *attack scenario*, where participants without programming skills should try to guess the correct results. Thus, half the participants of the non-programmers were encouraged to use any help for solving the tasks. Requirements for the tasks were a high success rate for programmers and a low success rate for non-programmers, even in an attack scenario. In addition, participants with programming skills should be able to solve the tasks relatively quickly. We had recommended a set of six tasks to be used for filtering non-programmers in online surveys. The most reliable screeners, however, were also those that took the most time.

In [60], we reported that the Qualtrics recruitment service charged for programmers 2.87€ per minute (43€ per 15min). Considering an exemplary sample of $n=100$ participants and a screening question of 4 minutes (the maximum a programmer needed in [12] for the code comprehension question), it would yield an additional cost of $100 \cdot 2.87 \cdot 4 = 1148€$ only for the screening question. As we suggest questions with a maximum time limit of 30 seconds (143€), the saving of the research budget is enormous (8 times the cost). While Qualtrics does not

As “programmer,” we refer to participants as defined in [60]: participants with programming skills from previous studies and participants who have worked with imperative programming languages (e.g., Java, C, Python)

Therefore, we investigated the research question: *Can the effectiveness of screening questions for non-programmers be improved by introducing time constraints?*

In this work, we developed a large set of tasks and tested different time constraints to improve the tasks’ performance in an attack scenario. We analyzed which tasks performed well in previous work, trying to improve on them. In addition, we included some tasks previously used as a filter in other computer science publications, as well as strong performing tasks from our previous work [60]. We explored the influence of different time constraints on the correctness rate for programmers and non-programmers in an attempt to improve the effectiveness of the tasks as a filtering tool. For this, we used three different time constraints: 1) prompting the participants to solve the tasks as quickly as possible, 2) requiring the participants to solve the tasks within a given time limit, and 3) using a control group without any time constraints.

We tested the tasks with computer science students to decide whether time constraints can be added without affecting the programmers’ task performance. In addition, we used this survey to filter out tasks with a low rate of correctness. We tested the remaining tasks with non-programmers recruited online in a timed attack scenario. We analyzed the influence of the time limit on non-programmers by comparing them to the non-programmers in our prior work [60]. Finally, we compared the success rate and the required time to solve the tasks with programmers in a count-down scenario. Based on the results, we recommend a set of well-performing tasks for researchers to be used in online surveys as a filtering device.

5.1.1. Baseline Study

This chapter expands on the work presented in Chapter 7, where they developed tasks to be used as a filter in an online survey. They created 16 questions testing different categories of programming knowledge and skills. They tested these tasks with 17 computer science students and 34 professional developers to prove participants with programming skills could solve them. Additionally, they analyzed the results of 50 non-computer science students and 50 participants from Clickworker [52] to test if non-developers can solve the tasks. The results showed that participants with programming skills could solve many tasks, while those without skills had significantly lower success rates. Furthermore, they looked at the results of 52 participants with programming skills mentioned in their Clickworker profile to see how the tasks performed in a scenario they were expected to be used. While the participants performed better than the participants without programming skills, they did significantly worse than the control group of computer science students and working software developers, again showing the need for a tool to filter out non-developers. Finally, we explored an attack scenario, where participants were given a monetary incentive of two euros per task to find correct solutions to the tasks. In the end, they recommended six tasks to be used as screening questions in surveys. For four of them, they suggested using time limits; however, without further exploring these. We built upon their work and investigate the usage of time limits for screener questions.

5.2. Methodology

In this section, we describe the methodology for the three conducted surveys in this work, the first one to estimate time constraints, the second one to test the screener questions, and the third one to simulate an attack scenario. Firstly, we present our screener questions in Section 5.2.1. Secondly, we discuss our study conditions in Section 5.2.2. Thirdly, we tested the questions with computer science students with different time constraints as described in Section 5.2.3.1. Finally, we simulated an attack scenario with Clickworker participants as described in Section 5.2.3.2. An attention check question [116] was placed randomly during the survey to filter out careless respondents (see Appendix C.1.1). After solving all the tasks, participants were asked if they had felt under pressure during the survey, and demographic information was collected.

5.2.1. Tasks

Our potential set of filter tasks consist of already established tasks from [60] and new tasks, which we developed to extend the question pool.

A list of all questions used can be found in Table 5.1, with the first six being the recommended tasks by us [60] and the newly developed tasks. The detailed questions with all the answer possibilities are additionally listed in the Appendix C.1.

Table 5.1.: Overview of all questions used for the timed scenarios

Baseline Questions from [60]		Abbreviation	Category
1	Which of these values would be the most fitting for a Boolean?	Boolean	Basic Knowledge
2	Choose the answer that best fits the description of a compiler's function	Compiler	Basic Knowledge
3	Choose the answer that best fits the definition of a recursive function	Recursive	Basic Knowledge
4	Please select the returned value of the pseudo code.	Backward.Loop	Code Comprehension
5	What is the purpose of the algorithm?	Sorting.Array	Code Comprehension
6	Which of these websites do you most frequently use as aid when programming?	Source.Usage	Information Sources
New Questions		Abbreviation	Category
1	Given the array arr[7,3,5,1,9], what could the command arr[3] return?	Array	Basic Knowledge
2	What is the parameter of the pseudo-code function?	Parameter	Basic Knowledge
3	Is the following pseudo-code function a recursive function?	RecursiveFunction1	Basic Knowledge
4	Is the following pseudo-code function a recursive function?	RecursiveFunction2	Basic Knowledge
5	Is the following pseudo-code function a recursive function?	RecursiveFunction3	Basic Knowledge
6	Is the following pseudo-code function a recursive function?	RecursiveFunction4	Basic Knowledge
7	What is the purpose of the pseudo-code algorithm above?	CountString	Code Comprehension
8	What is the purpose of the pseudo-code algorithm above?	Palindrome	Code Comprehension
9	What is the purpose of the pseudo-code algorithm above?	Prime	Code Comprehension
10	What would the pseudo-code return if i is 4?	Return	Code Comprehension
11	What issue would the following pseudo-code most likely cause?	Error.OutOfBounds	Finding Errors
12	What issue would the following pseudo-code most likely cause?	Error.Syntax	Finding Errors
13	When executing the pseudo-code below, your program never stops running.	Infinite.Loop	Finding Errors
14	Which of these IDEs have you used before?	IDE.Used	IDE Recognition
15	Please select 2 items from the list which are IDEs.	IDE.Known	IDE Recognition
16	Please select the two programming languages from the list below.	Rec.Lang	Programming language recognition

Baseline questions

Multiple questions (**Boolean**, **Compiler**, **Recursive**, **Backward.Loop**, **Sorting.Array** and **Source.Usage**.) from our survey were included to test if the participants' groups were similar enough to each other. Furthermore, even though some of these questions seemed to be solved with a relatively high

success rate in the attack scenario, where participants tried to guess the results, they may still be used as a filter based on the time to solve the tasks.

New questions

Based on the results of our previous surveys [60], a new set of questions was developed. We observed that participants from the attack scenario were able to google easy knowledge questions. Therefore, we aimed to test more comprehension questions. As the previous analysis of the attack scenario showed, non-programmer participants were less successful in using aid when tasks required them to read programming code. In addition, these tasks had a comparatively low success rate with non-programmers. **Palindrome**, **CountString** and **Prime** all asked participants to identify the purpose of a pseudo-code algorithm. In **Return**, participants were required to evaluate several if-statements in a function to predict the returned value. Additionally, **Recursive** was transformed so that participants did not only have to know the definition of a recursive function but also had to identify a recursive function if shown in **RecursiveFunction1-4**. This may increase the difficulty non-programmers have to solve the task.

Parameter was shown separately from **BackwardsLoop** on a new pseudo-code to ensure an independent result. Furthermore, several tasks were created where participants had to predict which error message a compiler would return if they tried to compile such code. One question, **Error.OutOfBounds**, where an array was accessed with a negative number, included two correct answers, as some programming languages allow negative indexing, resulting in undefined behavior, while others did not, resulting in an "Index out of range" error. Finally, two questions were included, asking participants to select IDEs they knew and IDEs they had worked with before. A similar task was used by Balebako et al. [28]. However, for this work it was changed into multiple-choice questions with some valid answers and some technical computer terms, as Balebako et al. stated that due to a large number of IDEs, the answers were sometimes hard to verify.

The order of all questions and answers was randomized. Only **RecursiveFunction1-4** were always shown together, however, again in randomized order.

Table 5.2.: Study conditions

Condition	Tested groups	Description
Base	Prog	No time constraints
No Limit	Prog	"Solve it as fast as possible", no specific limit given
Countdown	Prog, Attack	Time limit with countdown timer

Prog = Programmer, Attack = Non-Programmer Attack Scenario

5.2.2. Conditions

All conditions are summarized in Table 5.2. Participants with software development skills were split into three groups to evaluate the influence of different time constraints on the success rate. First, we used a control group without mentioning a time limit (**Base**). With this group, it could be determined if a time constraint had a negative impact on the correctness of the programmers'

solutions. Second, we established a group with a note before each question that the time to solve the next question would be measured and a request to solve the task as quickly as possible without mentioning an explicit time limit (**No Limit**). Participants were encouraged to work as quickly as possible to create the largest margin to the time potential non-programmer participants would need. Last, we provided a group of participants a time limit in the form of a countdown timer given for each question (**Countdown**). In contrast to the *No Limit* group, the *Countdown* group provides a clear cut-off point for passing the task and not passing the task.

To find an appropriate time limit for the group *Countdown* and to eliminate confusing questions, a test run with the *Base* group setup with four researchers of the security department was conducted. The time limit for the countdown group was based on the mean time of the test group, with an extra time of 50% or ten seconds, whichever was larger. The time limit was rounded to the nearest ten seconds.

Table 5.3.: Demographics of the Participants (n = 98)

	Base (n = 23)	Programmer (n = 74) No Limit (n = 26)	Countdown (n = 25)	Non-Programmer (n = 24) Countdown Attack (n = 24)
Gender	female: 5, male: 18	female: 6, male: 20	female: 6, male: 19	female: 10, male: 13, prefer not to tell: 1
Age	min: 20, max: 29, mean: 24.43, md: 25, sd: 2.21	min: 21, max: 31, mean: 24.38, md: 23, sd: 2.87	min: 22, max: 35, mean: 25.76, md: 25, sd: 3.13	min: 22, max: 60, mean: 34.67, md: 33, sd: 10.46
Main Occupation	Computer Science Student: 22, Full Stack Developer: 1	Computer Science Student: 22, Software Developer: 3, Python: 1,	Computer Science Student: 22, Economist: 1, System administrator: 1, Web Developer: 1	e.g., Teacher, Freelancer, Administrator, Business
Country of Residence	Germany: 22, Pakistan: 1	Germany: 25, Bangladesh: 1	Germany: 24, Armenia: 1	Germany: 7, USA: 6, UK: 3 and 6 Other Countries
General Development Experience [years]	min: 1, max: 15, mean: 6.65, md: 7, sd: 3.5	min: 1, max: 13, mean: 6.62, md: 6, sd: 3.26	min: 0.5, max: 10, mean: 5.58, md: 5, sd: 2.85	min: 0, max: 5, mean: 1.21, md: 1, sd: 1.47

5.2.3. Participants

To calculate the number of participants required to create statistically significant results, a power analysis was conducted with G*power [95]. As the results of the two independent groups were to be compared, the analysis was done for Fisher's exact test, and the proportion of correct answers of both groups had to be estimated. Since the results can be expected to be at worst similar to the ones in the attack scenario we conducted [60], the expected percentage was chosen to be the same, which was 61% for the non-programmer group. The programmer group worked on the same tasks, so 98% was selected as their rate of correctness. With p as 0.05, this leads to 24 participants required to show statistically significant results. A summary of the demographics for both programmer and non-programmer groups is available in Table 5.3.

5.2.3.1. Programmers: Computer Science Students

The participants of the programmers' group were students in a master's computer science class. All the participants attended at least one programming course during their Bachelor's program. Participation was not mandatory for passing the course, but participants received a small bonus on the admission to the exam for taking part in the survey. The study introduction for programmers is available in the Appendix C.1.4.

All in all, 77 participants took part in the study, and 75 completed the entire survey. The assignment to the different time groups was done at random and resulted in 24 participants in the *Base* group, 26 in the *Countdown* group, and 27 in the *No Limit* group. One participant in the *Base* group did not continue working on the survey after one task. Another participant in the *No Limit* group stopped halfway through the survey. Their results were not included in the analysis, leaving us with 23 participants slightly under-representing the *Base* group (see Section 5.2.3). In addition, one participant in the *Countdown* group was excluded as the time for each task was extremely low, often below two seconds. Of the remaining 74 participants, 57 were male and 17 were female. On average, participants were 25 years old and had six years of programming experience.

5.2.3.2. Non-programmers: Clickworker

Finally, a third survey simulating an attack scenario with a time limit was carried out. Here, participants were recruited on Clickworker.com [52] and given several tasks of the previous scenario with a time limit to solve each task. The results were compared to the results of the programmer group of the study with computer science students. The study introduction for the non-programmer group is available in the Appendix C.1.5.

All in all, 25 recruited participants finished the survey, and three dropped out during the survey. Of those 25, one participant failed to pass the attention check question and was excluded from the analysis. The remaining participants were, on average, 34.67 years old. The youngest participant was 22, and the oldest was 60 years old. Ten of the participants were female, 13 were male, and one did not answer the question. Some participants claimed to have worked in IT-related fields before. However, some participants may still have answered as a programmer would have during the demographics questions. As these results would only cause the effectiveness of the tasks to be underestimated, their results were still used for the analysis. In addition, participants were not excluded due to extremely low solving times, as guessing an answer might be a tactic employed by the participants to solve the tasks. The participants were given 2€ for completing the survey and again 2€ for each task they solved correctly. This matches our payments in [60]. With ten tasks providing bonus payment, this resulted in a maximum of 22€ per participant.

5.2.4. Statistical Analysis

When comparing the participants' results on the tasks, their answers are classified as "Correct" and "Incorrect." The study aimed at finding different success rates between programmers and non-programmers. Thus, the results were tested with Fisher's exact test (FET) for each question. In addition, the times required to solve single tasks or completing the entire survey were compared. As the results were rarely normally distributed, they were analyzed with the Wilcoxon rank-sum

test to detect statistically significant differences. As all tasks were compared individually, no Bonferroni-Holm correction was conducted.

5.2.5. Ethics

The institutional review board of the university approved the project. The participants of the study were provided with a consent form outlining the scope of the study and the data use and retention policies, and it was also complied with the General Data Protection Regulation (GDPR). The participants were informed of the practices used to process and store their data and notified that they could withdraw their data during or after the study without any consequences. The participants were asked to download the consent form for their own use and information.

5.2.6. Limitations

As the participants with programming experience were entirely recruited from computer science classes, some bias may have influenced the results of the tasks. This might have resulted in participants over-performing on some tasks if they covered something taught in detail at the university or underperforming if they were not. However, the topic was still common knowledge for professional programmers. There may have also been other factors influencing the results, as participants in different age groups or participants working in various fields may have produced different results on the tasks. A more extensive and more diverse group of participants with programming experience may enhance the accuracy of the results.

Furthermore, the participants recruited on Clickworker.com [52] may not have been an entirely representative population. In addition, the recruited non-programmer participants for the attack scenarios were only encouraged to simulate the targeted group, namely non-programmers who participate in programmer studies by accident or intentionally. This targeted group can not be recruited and identified easily, so their behavior is only approximated with a monetary incentive to solve the filtering tasks. In addition, there may have been participants in the non-programmer group who have had some programming experience, thus influencing the results. This would, however, cause the tasks only to under-report their filtering quality at worst.

Finally, only a limited number of tasks can be tested, with many possible suitable tasks remaining undiscovered. There is also no guarantee that the used tasks are optimal or that their performance will remain the same in the future if there are changes in the field of computer programming. In addition, the tasks only test if a participant has some programming skills and may not be suitable for researchers trying to recruit more specialized participants, for example, participants who are skilled in a particular programming language or who worked in a specific sub-field.

Due to an error, we only tested the question whether the participants felt pressured in general only for ten participants in the *Base* group. Unfortunately, this allows us only a limited comparison all three groups. Only a comparison between the *No limit* and *Countdown* group is possible.

5.3. Results

In this section, the studies with programmers and non-programmers are discussed. First, the tasks introduced in Table 5.1 were tested with computer science students. Participants were expected to

perform similarly to the participants who were previously classified as programmers in the baseline study [60]. The influence of the time constraints on the participants' performance was measured. In addition, the self-evaluation of the participants and its correlation to successfully solving the tasks was discussed. In addition, the pressure caused by the different time constraints was reviewed. Second, the results of the timed attack scenario with non-programmers were compared to the results of the programmers in this and the baseline studies.

5.3.1. Programmers

A summary of the time the participants required per task, and the percentage of correct answers is available in the Appendix C.1. It is worth noting that participants in the *Countdown* group occasionally selected an answer and then let the timer run out. These answers were still evaluated like all other answers and resulted in a correct solution if the answers were correct. Overall, the timer ran out 19 times, with six correct and six incorrect answers. Seven times no answer was given in time, which was evaluated as incorrect answers. The **Parameter** task was the one where participants ran out of time the most, resulting in six timeouts.

5.3.1.1. Ensuring Comparability with the Baseline Study

First, it had to be ensured that participants were similar in skill level to the programmers previously we recruited in the baseline study [60] so that the new tasks could be compared to the old ones. This was achieved by comparing the results of the programmer group from the baseline study with the results of the *Base* group of this study for the six baseline questions (see Table 5.1). In addition, the time taken to solve the tasks was compared for only four of the tasks, as participants in the baseline study solved two tasks on the same source code, and time was not recorded separately.

When comparing the correctness of the two groups with the Kolmogorov-Smirnov test, none of the tasks showed a statistically significant difference. Consequently, the data suggested that the skill level of the participants of our programmer group was close to the participants in our baseline study [60].

5.3.1.2. Comparison Between Groups

Figure 5.1 shows the number of correct solution by each programmer group (*Base*, *No Limit* and *Countdown*). For each task, the results of the three different groups were compared with Fisher's exact test (FET) to check for statistically significant differences between the groups. The test did not show a significant difference between the groups in any of the tasks.

In Figure 5.2, the time each group required to complete the entire survey is presented. Participants in the *No Limit* group completed the entire survey the fastest (min: 8.17m, md: 13.93m, mean: 14.78m, max: 31.25m, sd: 5.3m). The second fastest were the participants in the *Countdown* group (min: 8.35m, md: 18.75m, mean: 22.12m, max: 55m, sd: 13.30m). The time for one participant in the *Base* group and one in the *No Limit* group were filtered out, as they left the last page of the survey open after finishing it, thus having a drastically increased overall time. Participants in the *Base* group were the slowest (min: 7.68m, md: 17.77m, mean: 26.56m, max: 89.4m, sd: 21.96m).

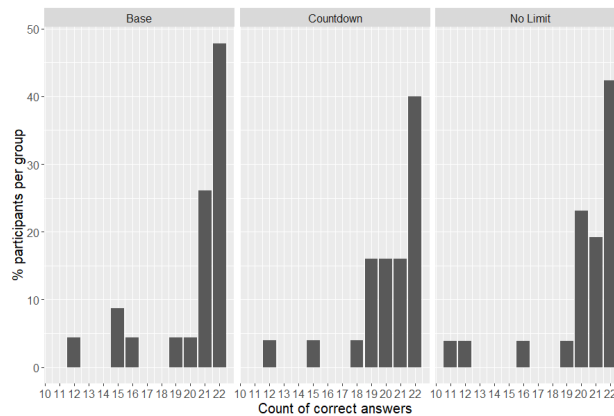


Figure 5.1.: Number of correct solution by each programmer group

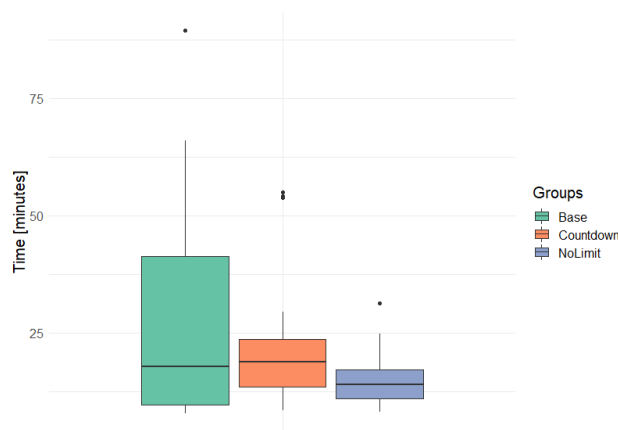


Figure 5.2.: Time required to complete the survey by each programmer group

The Wilcoxon rank-sum test did not show a statistically significant difference between the *No Limit* and the *Base* group (p -value = 0.39), and the *Base* and *Countdown* group (p -value = 0.73). However, between the *No Limit* and the *Countdown* group (p -value = 0.015*), a statistically significant difference was found. Noticeably, participants in the *Countdown* group were slower than the participants in the *No Limit* group, indicating that participants tended to spend more time on a task if they knew they have the time to do so.

5.3.1.3. Self Evaluation

All participants were asked to self-evaluate their programming skills after completing the tasks. Possible categories were "Beginner," "Intermediate," and "Expert." Of the 74 participants, 11 selected "Beginner," 54 "Intermediate," and 9 chose "Expert." Participants were not subdivided into the "Base," "No Limit," and "Countdown" groups.

For the following analysis, failing by overstepping the time limit was excluded as only participants from the time limit group could fail due to the limit.

Participants in the beginner group solved an average of 83% of the tasks correctly, about 18.2 out of 22 tasks. Participants who evaluated their skill level as intermediate solved 93% of the tasks correctly, with an average of 20.5 correct answers. In the expert group, 97% of all tasks were solved correctly by the participants, corresponding to 21.4 correct answers per participant on average. When comparing the groups with Fisher's exact test, there were significant differences between the beginner and the intermediate group (p-value < 0.0001) as well as between the beginner and the expert group (p-value < 0.0001). In addition, the difference between the intermediate and the expert group was significant (p-value = 0.024*). The results indicated that the self-evaluation of the participants might be used to predict the participants' success rate on the tasks.

5.3.1.4. Pressure

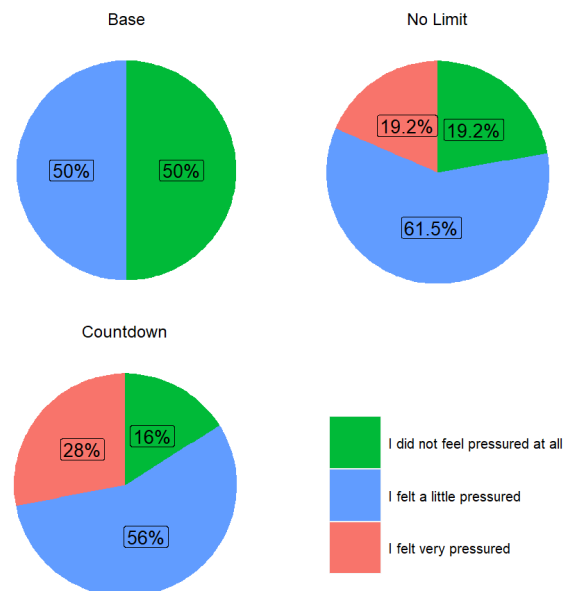


Figure 5.3.: Perceived pressure by each programmer group

Limitation: Since this question was added later to the survey, it was only shown to ten participants of the Base group. All other groups had the question already included in their survey.

At the end of the survey, participants in the *No Limit* and *Countdown* groups were asked if they had felt pressured by the time constraints during the survey. Additionally, *Base* participants were asked if they had felt pressured in general during the survey.¹ Three multiple choice answers were given, "I did not feel pressured at all," "I felt a little pressured," and "I felt very pressured." The results are shown in Figure 5.3.

¹Since this question was added later to the survey, it was only shown to ten participants of the *Base* group.

The data indicated that participants perceived pressure if a request to work fast or a strict time limit was given. The reported pressure is lowest in the base group. The Fisher's exact test did not show a statistical significance between the *Base* group and the *No Limit* group (p-value = 0.11). However, we found a statistical significant difference between the *Base* group and the *Countdown* group (p-value = 0.048*). Finally, there was no significant difference between the pressure felt by the *No Limit* group and the *Countdown* group (p-value = 0.86).

While the increase in pressure for participants is an unwanted side effect of the time constraints, it might reduce the risk of non-programmer participants using online services or help from others to solve the tasks. When participants are requested to work fast, time differences between programmers and non-programmers may increase, thus making it easier to differentiate between the two groups based on the time required to solve the tasks.

5.3.2. Non-programmers- Timed Attack Scenario

The attack scenario from our previous work [60] showed that by using a scenario with unlimited time constraints, non-programmer participants were able to solve the tasks with help. The correctness for programmers decreased significantly in neither the *No limit* nor the *Countdown* scenario. Thus, both scenarios might be suitable to filter out non-programmers. As the *Countdown* scenario delivers a clear cut-off point for all participants, it shows most clearly whether the participants could have solved the tasks within the time limit or not. In contrast, participants in the *No limit* group still could spend extra time double-checking their answers and thus increase the time required to solve the task above the cut-off point. For this reason, all tasks in the attack scenario were tested with a countdown timer, and non-programmer participants had the same time limit to solve each task as the programmers.

As participants with programming knowledge may be erroneously filtered out by tasks too difficult, only the ones with high success rates among programmers were included. For this, the rate of the correctness of all programmers' answers in the previous study was calculated. 95% was defined as the cut-off point, as it keeps the number of programmers failing the tasks low without expecting the programmers to give perfect answers. Thus, we included **Source.Usage**, **Recursive**, **Boolean**, **Rec.Lang**, **Compiler**, **RecursiveFunction3**, **RecursiveFunction4**, and **IDE.Known** as tasks in the survey. Additionally, **Array** and **Prime** were included, as they reach 95% if rounded to the closest whole number. **RecursiveFunction1**, which would fall in the latter category, was not chosen, as the correctness in the *Countdown* scenario was below 95% and a high success rate from guessing was to be expected in the attack scenario on the yes-or-no task. Finally, we considered ten tasks for the timed attack scenario with non-programmers.

5.3.2.1. Comparison to the Baseline Attack Scenario

As participants in the attack scenario we used in [60] worked on the tasks **Boolean**, **Compiler**, **Recursive**, and **Sources.Usage** without a time limit, we examined the influence of the limit on the success rate. The baseline participants without a time limit performed better on all four tasks. However, participants with a time limit were quicker on all four tasks, suggesting that our participants would have required additional time to solve the tasks to their satisfaction. It

strengthens the assumption that introducing a time limit decreases the chance of success for non-programmers on the tasks.

5.3.2.2. Comparison to Programmers

All in all, non-programmers needed, on average, 12.86 minutes (md: 10.09 minutes) to complete the entire survey. The fastest participant needed 3.88 minutes, whereas the most prolonged time taken for the survey was 27.47 minutes. The quickest participant solved all tasks correctly and noted in the comment section to work as a software engineer. We did exclude him from analysis because we were unable to prove this claim for correctness. In addition, the results could only cause the reported effectiveness to drop.

Participants got on average 3.71 (md: 3) correct answers. Since the goal was to distinguish the non-programmer participants from the programmers, their results were first compared to the computer science students' results. Here, only the results of the students in the *Countdown* group were considered, as the time constraint for both groups is the same.

The individual scores for each group are presented in Figure 5.4. On average, the non-programmers in the attack scenario solved 3.67 tasks correctly, with the lowest score being one and the highest being ten. For comparison, in the programmer group with the same countdown timer, two participants solved eight tasks correctly, another two nine, and the remaining 21 solved all the ten tasks correctly.

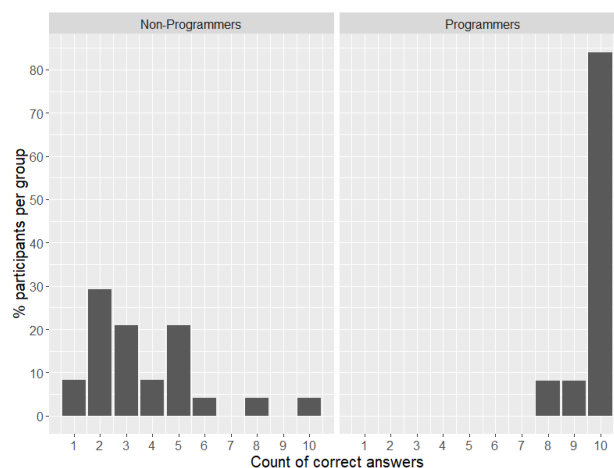


Figure 5.4.: Number of correct solutions by programmers and non-programmers

The results for both groups are presented in Table 5.4 for each task. The non-programmers performed worse than the programmer group on all tasks with regard to the success rate. In all but one task (the yes-or-no question **RecursiveFunction3**), the difference was statistically significant. In six of the ten tasks, the non-programmers were also significantly slower than the programmer group. The data shows that the correctness of non-programmers on the tasks significantly deteriorates with the introduction of a time limit, a phenomenon that did not occur when the same time limits were introduced to participants with programming skills. This indicates the usefulness of time

limits, as a correctly chosen limit prevents a significant number of non-programmers from solving a task. In contrast, the number of programmers solving the task correctly remains unchanged.

5.3.2.3. Aid Used

The non-programmer participants were asked if they used any form of outside help to solve the tasks. Possible answers covered "I knew the answer," "I guessed the answer," "Asked friends/colleagues," "Internet search (if possible, insert the link you used)" and "Other." For the internet search and "Other," participants could provide additional information in a text box. Due to the fact that participants only selected "Other" twice, stating that they had run out of time, it was not analyzed in detail. Since participants could use multiple different sources of aid, multiple selections were possible. The most common practice used was to guess the answer, which was done 46% of the time. Participants claimed to have known the answer 32% time, did an online search in 15% of the cases, and asked their friends or colleagues in only 7% of the cases.

In addition, we compared the frequency of the aid usage with the baseline attack scenario we conducted in [60]. When comparing all aid used during the entire surveys of both groups, the percentage of the different used support differed significantly ($p < 0.0001$, FET). In the untimed baseline scenario, participants knew or guessed the answer in 84% of the cases, an online search was done in 26% of the cases, and friends or colleagues were asked in 11.84% of the cases. The data showed that participants could still significantly improve their correctness by conducting an online search. With the time limit, they tended to try it less by a significant margin.

5.4. Discussion and Recommendations

Participants taking part in programming studies without the according skills can significantly threaten the validity of the data. Screener questions testing programming skills can be used to validate the target group. However, programmers' time is valuable and should not be wasted on screener tasks taking much time. Our analysis indicated that the time required to solve screener tasks could be reduced by using time constraints without decreasing the rate of correctness. Participants without programming skills performed significantly worse on the timed tasks both compared to our programmers and the baseline non-programmers without a time limit from [60]. Our non-programmers searched the Web less frequently and had to guess or trust their knowledge more often than the baseline attack scenario group without a time limit. These results showed that the introduction of the time limit and the new suggested screener questions helped decrease the success rate of non-programmers, thus improving the tasks introduced in [60] for filtering out non-programmers.

In the following, the setup of the tasks as a filtering instrument for online surveys is discussed. First, we outline which tasks we would recommend for research surveys with programmers. Second, we argue for the usage of a time limit for the tasks. Finally, we discuss how many of the tasks we would recommend using in a survey.

5.4.1. Screener Tasks

An essential characteristic of the tasks is excluding a critical number of programmers from the survey. With the previous cut-off point of 95% correctness for programmers, this criterion was fulfilled.

Additionally, the tasks must filter out a large number of non-programmer participants. As the tasks **RecursiveFunction3** and **RecursiveFunction4** are yes-no questions and increase the chance of guessing the answer correctly to a significant extent, the correctness of answers given by non-programmers is comparatively high. While it may still be possible to filter with a larger number of yes-no questions, the time to solve these tasks and the programmers' correctness rate does not differ fundamentally for the other tasks. Thus, we would not recommend them for the instrument. Further, the tasks **Compiler** and **Rec.Lang** both have a success rate of over 40%, indicating that a larger number of non-programmers are familiar with the terms or can look them up and answer these questions quickly. Thus, we also recommend excluding them from the instrument.

The remaining six tasks **Array**, **Prime**, **Boolean**, **IDE.Known**, **Recursive** and **Sources.Usage** reduce the success rate of non-programmers to 33% or lower. It should be noted, however, that the task **Prime** has a timer of three minutes and might take too long, especially if the explored survey is rather short. The recommended tasks are summarized in Table 5.5.

5.4.2. Time Constraints

As shown during the test with programmers, the timer with a countdown did not impact correctness. However, it affected non-programmers' results concerning their correctness. In addition, the participants used less often help to solve the tasks when being presented with a time limit, thus again worsening their results. For these reasons, it is beneficial to add a countdown timer to the tasks when filtering out non-programmers. Nevertheless, participants should be informed about the time limit before and during the tasks to avoid confusion with the conditions required to participate in the survey.

5.4.3. Setup of the Instrument

Some possible setups of the instrument are discussed here as a recommendation. While researchers are free to choose and discard tasks to their liking, it is advisable to randomize which tasks are shown if participants are likely to communicate and share their solutions. In addition, presenting multiple tasks can improve the instrument's accuracy, with the consequence that it takes longer to solve the tasks.

For the following calculations, the average success rate for all six tasks is used for programmers and non-programmers. To simulate the picking of the tasks at random, each task will leave 25% of the non-programmer and 98% of the programmers in the study.

One correct out of one: Showing participants one of the six tasks at random is expected to eliminate 75% of the non-programmers while only excluding about 1.78% of actual programmers. While this setup takes the least time participants spend on the filtering questions, more non-programmers could be identified if multiple tasks were combined.

Two correct out of two: In this scenario, potential participants would be given two of the six tasks and would have to solve both correctly in order to qualify for being accepted in the study. While this setup takes longer for participants, as they have to solve two tasks, it allows a larger number of non-programmers to be filtered out, approximately 94%. However, this setup also causes more real programmers to be filtered out, namely 3.53%. This setup is the most effective to filter out non-programmers.

Two correct out of three: Here, participants would be shown three of the six tasks and would have to solve two of them correctly to proceed in the survey. While not fewer non-programmers than in the previous setup would be eliminated, most programmers would be kept in the study. Only 0.09% of the programmers would be wrongly filtered out. However, a larger portion of non-programmers would remain, only approximately 84.37% being filtered out. In addition, this setup would take longest for the participants to complete.

Our analysis showed that researchers should use filter methods if there is any doubt on the participants' programming skills. We presented six screener questions that could be used either all or selected for research surveys with programmers. We suggest using either one, two, or three questions in a combination, with a preference for the second option. The exact method, however, should be chosen according to the researchers' requirements.

5.5. Summary

To ensure data quality, researchers should screen out non-programmers from online surveys requiring programming skills. In this chapter, we tested existing screener questions we suggested in [60] for programmers with different time constraints. Further, we improved and extended the screener questions to increase the difficulty for non-programmers to solve them, even if participants were using additional help. The new tasks and some previously used tasks were tested with programmers. The results showed that the participants were able to solve a high number of the tasks reliably and that the created time limits did not affect the success rate of the programmers.

Ten new and old tasks yielding the best results were tested with a time limit in an attack scenario with non-programmers. The participants were given a monetary bonus for each task solved correctly to simulate an attack scenario within a time limit for each task. Results proved that participants had difficulty employing aid from different sources when a time limit was given, resulting in an overall low correctness rate for several tasks.

In the end, six tasks with the best performance were recommended and different setups for the tasks in the survey were discussed. Thus, researchers can eliminate most non-programmers from their surveys reliably and effectively without simultaneously excluding a large number of their target programmer group.

Table 5.4.: Comparison of programmers and attack scenario

Group	Task	Minimum	Median	Mean	Maximum	Correctness
Countdown Programmer Countdown Attack	Boolean	4.3s	9.77s	11.76s	25.45s	100
		4.45s	16.65s	19.44s	42.16s	25
Countdown Programmer Countdown Attack	Recursive	6.32s	11.24s	13.28s	30.01s	100
		5.31s	19.6s	19.49s	32.24s	29.17
Countdown Programmer Countdown Attack	Sources.Usage	6.08s	9.64s	10.53s	22.26s	100
		4.36s	16.73s	18.73s	30.17s	33.33
Countdown Programmer Countdown Attack	Rec.Lang	5.02s	10.16s	11.64s	21.05s	100
		6.68s	19.08s	19.8s	31.85s	41.67
Countdown Programmer Countdown Attack	Compiler	6s	12.94s	16.5s	40.01s	100
		1.64s	23.16s	21.05s	40.11s	45.83
Countdown Programmer Countdown Attack	Array	8.67s	65.91s	86.70s	180.12s	96
		5.61s	32.03s	32.61s	60.02s	16.67
Countdown Programmer Countdown Attack	Prime	16.7s	49.73s	62.93s	165.35s	96
		4.46s	56.69s	68.04s	169.42s	20.83
Countdown Programmer Countdown Attack	IDE.Known	5.89s	14.57s	18.65s	60.21s	96
		5.08s	28.57s	33.11s	60.14s	25
Countdown Programmer Countdown Attack	RecursiveFunction4	5.42s	19.19s	23.93s	64.13s	96
		2.6s	24.8s	31.54s	82.95s	50
Countdown Programmer Countdown Attack	RecursiveFunction3	4.81s	11.4s	15.51s	42.56s	92
		4.03s	23.03s	29.71s	90.01s	79.17

The tasks are sorted descended by correctness of the programmer group and ascended by the correctness of the countdown attack group.

Table 5.5.: Overview of recommended screener questions

Abbreviation	Screener Question	Time Limit	Correctness Programmer	Correctness Non-Programmers
1 Sources.Usage	Which of these websites do you most frequently use as aid when programming?	30s	100%	33.33%
2 Recursive	Choose the answer that best fits the definition of a recursive function	30s	100%	29.17%
3 Boolean	Which of these values would be the most fitting for a Boolean?	30s	100%	25%
4 IDE.Known	Please select 2 items from the list which are IDEs.	30s	96%	25%
5 Prime	What is the purpose of the pseudo-code algorithm above?	3m	96%	20.83%
6 Array	Given the array <code>arr[7,3,5,1,9]</code> , what could the command <code>arr[3]</code> return? The array starts with 0.	1.5m	96%	16.67%

6. Deception: Study Announcement in a Password-Storage Study

Disclaimer: The contents of this chapter were published as parts of the publication "Replication: On the Ecological Validity of Online Security Developer Studies: Exploring Deception in a Password-Storage Study with Freelancers", presented at the 16th USENIX Symposium on Usable Privacy and Security (SOUPS'20) [58]. The concept and study design were developed by Matthew Smith, Alena Naiakshina, and me. I conducted the statistic analysis. Johanna and I evaluated the qualitative data. The study was set up and conducted online by Johanna Deuter and Alena Naiakshina. I discussed the key insights with Alena Naiakshina and Matthew Smith. I wrote the paper in cooperation with Alena Naiakshina, Johanna Deuter and Matthew Smith.

6.1. Motivation

One major issue with software developers is their limited availability as subjects for research studies. Professional developers often cannot set aside time, may not be locally available or have hourly rates that researchers cannot afford [12, 117, 161, 11, 14, 114, 192]. An option to recruit professionals for studies is cooperating with companies as can be seen in [45, 63]. However, it can be difficult to find companies willing to join such studies. Reasons we have encountered are, that companies are hesitant to allocate time of their developers to a study, both for time and cost reasons, as well as worries about disclosure of information about their company as part of the publication process. Therefore, many previous security developer studies were conducted with computer science (CS) students [130, 131, 11, 30, 135, 192] or developers recruited on GitHub [14, 94, 156, 185, 186]. To supplement these recruitment options, Naiakshina et al. [132] proposed to use platforms such as *Freelancer.com* for developer recruitment.

In 2017 and 2018, Naiakshina et al. [130, 131] conducted a qualitative and a quantitative password-storage study with CS students to provide more insights into developer's security behavior. The participants were asked to complete the user registration functionally of a university social networking platform. Half the participants were *prompted* for secure password storage in the task, while the other half were merely told the study is about API usability (*non-prompted*). The results showed that not a single participant stored user passwords securely without being prompted. Some students, however, noted that they would have saved the user passwords securely if they had been working on a project for a real company. In order to find out whether the previous results were a study artifact, Naiakshina et al. conducted a follow-up study with freelancers [132], which they did not announce as an academic study but as a real project. In the corresponding pilot study, the

	CS Students [130, 131]	Freelancers [132]	This Study: Replication of [132] with Freelancers
Independent Variables (IV)	IV1: Security prompting (yes/no) IV2: Framework (JSF/Spring)	IV1: Security prompting (yes/no) IV2: Payment (100/200 euros)	IV1: Security prompting (yes/no) IV2: Payment (100/200 euros) IV3: Framework (JSF/Spring)
Study Context and Task	University researchers University social networking platform	Start-up Sports photo social networking platform	University researchers Sports photo social networking platform
Study Deception	No	Yes	No
Recruitment	University	Individually on Freelancer.com	Project on Freelancer.com
Post Security Request	No	SecRequest-P if plain text submission	SecRequest-P if plain text submission SecRequest-G if not implemented industry security standards

Table 6.1.: Overview of password-storage studies for most relevant aspects.

authors found that owing to the university context of task from the previous study, freelancers believed that they were working on university homework and also did not implement any security. To avoid this potential study bias, the authors invested additional study design work to provide a more realistic scenario to make the freelancers believe they were working for a real company and the code would be used in the wild. First, they removed the university social network context and posed as a start-up that had just lost a developer in their team and was searching for a new one. The job freelancers were asked to complete was to create code for a social networking platform for a sports photo sharing website. To make the scenario more believable, the researchers created a fake multi-page online web presence for the start-up. Second, the authors had to manually contact freelancers individually, instead of advertising the job on the service, and letting the freelancers apply for the job. The latter would have made recruitment easier, however, the Freelancer.com platform shows who has been hired for projects advertised in this way and pilot studies showed high dropout rates. This was due to freelancers being confused and suspicious about a single job being given to many freelancers. However, using the deception study design, Naiakshina et al. concluded that the CS students and the freelancers behaved similarly with regard to their password storage practices in this particular study setup and thus, suggested Freelancer.com as a promising platform for developer recruitment which warrants further examination.

In this work, we specifically want to examine the use of deception in the context of this study. Deception can be a useful tool, if disclosing the study purpose would lead to a significant change in behavior in the participants, e.g., Naiakshina et al. were concerned that freelancers would not implement security in the same way in a study setting as they would for a real customer. However, deception should be used only after careful consideration. Three issues are relevant in our context 1) One of the fundamental principles of human subjects research is *informed consent* and deception can impact this principle. 2) If deception is used, participants who take part in multiple studies or have heard of deceptive studies might second guess what the study is about, potentially affecting their behavior in an unknown manner. 3) Deception studies can require additional study design work, e.g., in the case of Naiakshina et al. the creation of a fake company web presence and more elaborate scenarios.

In order to gain more insights into study methodology and ecological validity of developer studies, we replicated the freelancer study of Naiakshina et al. [132] except for the use of deception. While the study task was kept the same, we created a new profile on Freelancer.com, where we introduced ourselves as a university group conducting scientific research and the job as a study. Due to this we were able to post and manage our study as a single project and manage all freelancers from there, reducing the additional study design work needed to run the study.

In addition to examining the effect of deception, we add further insights into the effect of different application programming interfaces (API) offering cryptographic libraries. For this we replicated the comparison of two frameworks JSF and Spring, as examined in the studies [130, 131],

Finally, we investigated whether providing participants with password storage guidelines has an effect on the security of the submitted solutions. Table 6.1 provides an overview of the previous lab studies with computer science students [130, 131], the freelancer study [132], and the present study.

6.2. Methodology

With the aim of improving the ecological validity of their study, Naiakshina et al. [132] concealed the context of their scientific research and hired freelancers for a “real project.” The authors invested additional study design work into the deception by creating a fake start-up with a web presence, creating a fake profile on Freelancer.com, designing a task description as authentic as possible and contacting and hiring individual developers based on their skills.

To test whether similar results are achieved without the use of deception, we conducted a replication-extension [36] of Naiakshina et al.’s [132] freelancer study, which adopted the study design from the original CS student studies [130, 131]. While we replicated the study of Naiakshina et al. with freelancers [132], we also extended it by the methodology (study announcement) and the framework variable, which we adopted from the previous student studies [130, 131] as well as an additional security request (see Table 6.1). The task was to complete a Java registration functionality that facilitated the storage of user properties (including user passwords) from a web form in a database. We used the task description given in [132] and hired participants from Freelancer.com. Similar to the previous studies, *prompting* was one variable in our study: half of the participants were tasked to securely store the passwords while the other half was not explicitly asked to do so.

When the project was published on Freelancer.com, the participants bid on it, and we contacted them via private messages. To communicate with the participants, we used the playbook from [132] and extended it when new cases appeared. The changes we made are given in the Appendix D.5. After completing the programming task, participants were asked to fill out an online survey to obtain their opinions on the used frameworks, their demographics, and their feedback about the task. In the following section, we provide a detailed description of all the design changes we made in contrast to the previous study [132].

6.2.1. Study Design Changes

Recruitment. One major change from the previous work was the public posting of the project on the freelancer platform as part of a scientific study. In the freelancer study of Naiakshina et al. [132], the researchers sent private messages with the project offer to freelancers who had mentioned Java knowledge in their profiles. Due to the limited filter features, the researchers needed to manually inspect freelancers’ profiles to verify whether they actually had the required knowledge. Eighty of the 340 selected subjects did not have the required knowledge. Additionally, the remaining 260 freelancers were contacted by the researchers, but a total of 211 did not accept the offer for different reasons, such as their lack of experience or time. In our study, developers had to submit an application for the project and thus, it was ensured that all the applicants were available for the study. Our public announcement for the study is available in the Appendix D.5.1.

Study Announcement. In the previous study, the researchers presented themselves as a start-up company and revealed their academic aims only at the end of the programming task. To make the project as realistic as possible, the original task presented in [130, 131], needed to be changed from a university setting to a company setting. The authors created a fake company profile on Freelancer.com and a web presence for that company and shared that with the subjects. While we used the same task description as used in the previous freelancer study, we omitted all the other steps and introduced ourselves as academic researchers conducting a scientific study.

Framework. While in the original freelancer study only JSF was used, we randomly assigned the participants to use either JSF or Spring as introduced in the previous student studies [130, 131]. The JSF participants had to implement secure password storage on their own. In contrast, Spring offers supporting libraries.

Security Requests. In their CS student studies, Naiakshina et al. [130, 131] accepted the participants' initial solutions and evaluated them based on a security score. To sum up, participants received a score of 0-7 points for security, based on their implementation choices for the password storage security, such as the hashing algorithm, salt generation, iterations etc. (see Section 6.2.4). In the freelancer study, the authors extended the security score and included a security request:

- *SecRequest-P(laintext)*: If the submissions included plain text password storage, subjects were asked to revise their submissions and to securely store user passwords.

We adopted this procedure and extended it by a further security request:

- *SecRequest-G(guideline)*: If the security score of participants' solutions was less than 6 points, we asked them to store the passwords by following industry's best practices.

As in the previous studies by Naiakshina et al., we accepted 6 points, instead of the full 7, for security, because Springs' default implementation of the bcrypt scores 6 points; thus, no one received the full 7 points by using memory-hard hashing functions. To investigate whether developers could obtain the full 7 points when given the appropriate source, we provided our participants with website links to the password security guidelines of the Open Web Application Security Project (OWASP) [3] and National Institute of Standards and Technology (NIST) [96] for the SecRequest-G. The exact wording and an illustrated procedure of the security requests can be found in the Appendix D.1.

Compensation. In the freelancer study, Naiakshina et al. [132] tested the impact of a payment variable on security by recruiting subjects either with a payment of €100 or €200. After revealing the study context, freelancers were invited to fill out a survey for additional €20. To allow researchers to conduct scientific studies in an economical manner, we began recruiting participants with a lower compensation amount of €120 for the project, including the programming task and the survey. Naiakshina et al. did not find a significant effect between the two payment levels on security. Therefore, after facing difficulties to recruit over 18 participants with €120, we started offering €220.¹

Performance Based Payment. Based on the insights obtained from our pilot study (see Section 6.2.2), our payment method resembled the prior freelancer password storage study [132] but was split into compensation milestones. Since we included up to three possible iterations of the code review and possible security requests, we divided the payment process based on three milestones: €50 for the first code submission, €50 for the final code release after our review including possible security requests, and €20 the survey completion. Accordingly, participants received €100, €100, and €20 respectively for each milestone for a payment of €220.

¹As in the previous freelancer study [132], payment did not show an effect on the decision to include security in the initial solution (FET: $p = 0.55$, OR = 1.54, CI = [0.39, 6.19]). We also regarded the prompting vs. the non-prompting group and did not find any significant effect in both cases.

6.2.2. Pilot Study

We recruited two freelancers via private messages to participate in our pilot study. We offered €220 to each. One participant reported that he wanted further instructions instead of just the mention of OWASP and NIST. Therefore, we included the links to the security guidelines in the final study. Since one participant appeared to be bothered by the requests, we divided the payment process based on milestones to clearly indicate that there were additional steps in our study. One participant stated that it took him six hours to finish the task, but he submitted the solution after six days. The other participant worked for three and a half hours on the task. Since both the participants reported to have finished the task in a relatively short period of time we started the recruitment process with €120.

General Information:			
Gender	Male: 32	Female: 11	NA: 0
Age	min: 18, max: 46	mean: 28.95, median: 29	SD: 6.21
Country of Residence	China: 11, Hong Kong: 1, The Netherlands: 1, Palestine: 1,	India: 10, Gaza: 1, New Zealand: 1, Romania: 1,	Pakistan: 6, Russia: 3, Lithuania: 1, Vietnam: 1, Germany: 1, Malaysia: 1
Profession and Programming Experience:			
Profession	Freelance Developer: 28 Employed full time: 1	Industrial Developer: 12 Undergraduate full-time Student: 1	Finance Professional: 1
General Development Experience [years]*	min: 1, max: 20	mean: 7.42, median: 7	SD: 4.47
Java Experience [years]*	min: 1, max: 16	mean: 6.19, median: 5	SD: 3.69

* = There were no significant demographic differences between the groups.

Table 6.2.: Demographics of participants (n = 43)

6.2.3. Participants

Recruitment. We posted our project in 5 iterations over a time-span of almost two months. On each public project, freelancers could place bids with their payment offer, which ranged from €120/€220 to €250. In total we received 101 applications, of whom we invited 73 to the study. Since we limited our payment to €220, participants with higher bids were not invited to the study. In total, we excluded 28 applicants for requirement reasons, such as participants with a bid higher than we offered (12), already participated in our study (9), and other reasons (7), such as missing Java skills in their profile or large time frames. Forty-three accepted our study invitation and participated in the study.

The first 3 iterations were posted with a payment of €120 and received a total of 60 bids from which we invited 46 freelancers to the study. Fourteen were removed for requirement reasons. Eight participants did not answer to the study invitation and another 8 were not interested anymore after reviewing the study material. Two wanted a higher payment for the project, one had no Java skills, two declined without seeing the study materials and one did not believe that we were conducting a study. Finally, 24 agreed to participate in our study. In the last 2 iterations, we offered a payment of €220. We received 41 bids and contacted 27 potential participants, of which 19 agreed to participate in the study. Fourteen freelancers were removed for requirement reasons. Six participants did not respond and one told us he is not interested anymore. Another participant wanted a higher payment for the project. In total 43 freelancers participated in our study and were randomly assigned to one of the 4 conditions: Spring-Prompting (FSP), Spring-Non-Prompting (FSN), JSF-Prompting (FJP), and JSF-Non-Prompting (FJN).

Demographics. Table 6.2 summarizes the demographics of our participants. While the demographics were comparable to Naiakshina et al.'s [132] freelancers in general, more female participants were involved in this study. 74% (32 of 43) were male, and 26% (11 of 43) were female. Most of the participants claimed to be from China (11), India (10), and Pakistan (6). Their ages ranged between 18 and 46 years (mean: 28.95, median: 29, SD: 6.21). The general programming experience of the participants ranged from 1 to 20 years (mean: 7.42, median: 7, SD: 4.47). For Java, the programming experience was reported to be between 1 and 16 years (mean: 6.19, median: 5, SD: 3.69). Almost all the participants reported to be experienced in developing web applications (41 of 43) and desktop applications (27 of 43). Thirty-eight participants reported to have a university degree. The freelancers had the option of indicating a minimum hourly wage in their Freelancer.com profile. The lowest rate among our participants was €5/hour, while the highest was €46/hour (mean: 21.71, median: 19, SD: 11.3, NA: 2).

6.2.4. Evaluation

Code Analysis. When releasing the milestones, we accepted only functional solutions. We adopted the extended version of the security scale [130] for the password storage security used by Naiakshina et al. [130, 131, 132] to score the code submissions. To sum up, we used a binary variable *secure* that indicated whether a participant included at least some kind of security. An ordinal variable *security score* was used to assess the security of the solution according to the password security scale [130] from 0-7; the *security score* considered the hash algorithm, iteration count for key stretching and salt generation. Submissions in which the user passwords were stored as plain text in the database received 0 points. Base64 and symmetric encryption are not suitable methods for secure password storage, and thus, any submissions that included these methods were rated being insecure (0 points). The security scale can be found in the Appendix D.2.

Two coders independently reviewed all the programming code submissions and evaluated them for security. Disagreements were resolved by consulting a security expert and discussing the algorithm specifications. We had 7 cases of disagreement, e.g., if one coder assessed the iterations incorrectly or misread the hash length. However, after a discussion all cases were resolved. With the rigid scoring system and the strict algorithm specifications, full agreement was achieved among the researchers.

Statistical Testing. We evaluated the same hypotheses as Naiakshina et al. [131, 132]. We were able to examine the effect of prompting vs. non-prompting (H-P1), framework (H-F1), years of Java experience (H-G1), and password storage experience (H-G2) on security. We also used the same statistical tests as in [131, 132]. All the tests on the same dependent variable were corrected using the Bonferroni-Holm correction. We denoted all corrected tests with "family = N," where N is the family size, and reported both the initial and corrected p-values (cor-p). An additional description of the hypotheses and a summary of our statistical analysis can be found in the Appendix D.3 and D.4.

Qualitative Analysis. The open-ended questions from the follow-up survey were analyzed by two researchers using inductive coding [171]. The two researchers independently searched for codes categories and themes emerging in the raw data. The codes were compared and the inter-coder agreement was calculated by using the Cohen's kappa coefficient (κ) [53]. The agreement

was 0.83. A value above 0.75 is considered a good level of coding agreement [87]. We found a large number of similar codes as in the previous freelancer study [132]. In order to provide novel insights, we only report new codes and findings in this work.

6.3. Limitations

Sample. Similar to Naiakshina et al. [132], our sample consisted of developers from Freelancer.com. This sample is not representative for all developers. Other freelancer hiring services could be used by other developers, and the results may vary. Further, since we publicly announced the project, participants needed to actively contact us, leading to a possible self-selection bias.

Recruitment Payment. We used two payment levels to recruit participants. This might have led to a self-selection bias. However, we tested the effect of payment on security in the initial solution and did not find any significant effect between both payment levels.

Deception. Similar to Naiakshina et al.'s previous password storage studies, this work examined the prompting vs. non-prompting effect in the task description. This resulted in concealing the security-focused research from half of our participants. However, due to our security requests, participants were able to improve their submissions. We received only positive feedback from our participants.

Generalizability. Finally, our findings are based on a single example study and thus further studies are needed to see if our results replicate in other types of studies.

6.4. Ethics

The institutional review board of our university approved our project. The participants were sent a link to a consent form in the first message of the conversation. We informed them of our data-storage policies and that they could withdraw their data at any time. To treat all the participants equally, we compensated the participants who had initially been offered a lower pay amount (€120) with additional €100 at the end of our study. Thus, all our participants received €220 for their efforts.

6.5. Results

In this section, we present an analysis of the present study. Additionally, we compare the results of the present study with results from the previous studies [131, 132]. To enable this comparison, we investigated the effect of the same factors on whether participants decided to store user passwords securely in the database considering the initial submissions. Further, we compared submissions from the previous freelancer sample [132] with our sample.

6.5.1. Security

As in the previous freelancer study [132], our participants used three techniques to store user passwords in the database: (1) hashing (+ salting); (2) symmetric encryption; and (3) Base64 encoding. We rated the solutions according to the security scale introduced in Section 6.2.4.

Table 6.3 shows the summary of the initial password storage solutions and the solutions handed in after SecRequest-P (in bold). To submit their initial solutions, participants took on average 4 days (min: 2h 20 min, max: 29 days, median: 2 days, SD: 5.4 days). In total we received 23 non-secure and

Participant	Prompting	Framework	Payment	Working Time	Include SecRequest-P	Active Working	Security Requests	Function	Length in bits	Iteration	Salt	Secure	Score	Copied
FJN1	0	JSF	120	3 Days	3 Days	7h	P + G	SHA-1	160	1		0/1	0/2	
FJN2	0	JSF	120	3 Days	2 Days	4h	P	bcrypt	184	2 ¹⁰	SR	0/1	0/6	
FJN3	0	JSF	120	18 Days	2h 30min	11h	P + G	MD5	128	1		0/1	0/1	✓
FJN4	0	JSF	120	3 Days		8h		PBKDF2 (SHA-512)	512	20 000	SR	1	6	
FJN5	0	JSF	220	2 Days	30min	12h	P + G	MD5	128	1		0/1	0/1	
FJN7	0	JSF	120	3 Days		25h	G	MD5	128	1		1	1	
FJN8	0	JSF	120	7 Days		4h	G	PBKDF2 (SHA-512)	512	65 536	St	1	5	
FJN9	0	JSF	220	12 Days	1 Day	30h	P + G	SHA-1	160	1		0/1	0/2	
FJN10	0	JSF	220	3 Days		8h	G	pgCrypto(xdes)	64	725	pgC	1	3	
FJN11	0	JSF	220	2h 20min		4h		bcrypt	184	2 ¹⁰	SR	1	6	
FJN12	0	JSF	220	2 Days	35min	8h	P + G	MD5	128	1		0/1	0/1	
FJP1	1	JSF	220	1 Day		5h		bcrypt	184	2 ¹⁰	SR	1	6	
FJP2	1	JSF	120	3 Days		48h	G	sym. Encryption				0	0	
FJP3	1	JSF	120	1 Day		10h	G	PBKDF2 (SHA-1)	128	65 536	SR	1	5	
FJP4	1	JSF	220	1 Day		18h		bcrypt	184	2 ¹⁴	MR	1	6	✓
FJP5	1	JSF	120	6 Days		16h		PBKDF2 (SHA-1)	160	20 000	SR	1	6	✓
FJP6	1	JSF	120	2 Days		3h	G	MD5	128	1		1	1	✓
FJP7	1	JSF	120	3 Days	15min	10h	P + G	MD5	128	1		0/1	0/1	✓
FJP8	1	JSF	120	4h		1h	G	MD5	128	1		1	1	✓
FJP9	1	JSF	120	12 Days		60h	G	Base64				0	0	
FJP10	1	JSF	120	1 Day		12h	G	sym. Encryption				0	0	
FJP11	1	JSF	220	2 Days		10h	G	sym. Encryption				0	0	
FSN1	0	Spring	120	3 Days	5h	15h	P + G	SHA-512	512	1	SR	0/1	0/5	
FSN2	0	Spring	220	8h 30min	20min	4h	P	bcrypt	184	2 ¹⁰	SR	0/1	0/6	
FSN3	0	Spring	120	7 Days	25min	10h	P	MD5	128	1		0/1	0/1	
FSN5	0	Spring	120	1 Day		6h	G	sym. Encryption				0	0	
FSN6	0	Spring	120	13 Days	19 Days	30h	P	bcrypt	184	2 ¹²	SR	0/1	0/6	
FSN7	0	Spring	120	1 Day	1 Day	10h	P + G	SHA-1	160	1		0/1	0/2	
FSN9	0	Spring	220	1 Day	2h	6h	P + G	SHA-256	256	1		0/1	0/2	
FSN10	0	Spring	120	1 Day	1h 10min	6h	P + G	sym. Encryption				0/0	0/0	
FSN11	0	Spring	220	9h	20min	2h	P	bcrypt	184	2 ¹⁰	SR	0/1	0/6	
FSN12	0	Spring	220	1 Day	3 Days	9h	P	bcrypt	184	2 ¹⁰	SR	0/1	0/6	
FSP1	1	Spring	220	29 Days		72h		bcrypt	184	2 ¹⁰	SR	1	6	
FSP2	1	Spring	120	4 Days		20h	G	SHA-256	256	1		1	2	
FSP3	1	Spring	120	4h 30min		2h		bcrypt	184	2 ¹⁰	SR	1	6	
FSP4	1	Spring	120	5 Days		10h	G	MD5	128	1 000	SR	1	4	
FSP5	1	Spring	220	2 Days		16h		bcrypt	184	2 ¹⁰	SR	1	6	
FSP6	1	Spring	120	1 Day		12h	G	Base64				0	0	
FSP7	1	Spring	220	7 Days		20h		bcrypt	184	2 ¹⁰	SR	1	6	
FSP9	1	Spring	220	1 Day		11h		bcrypt	184	2 ¹⁰	SR	1	6	
FSP10	1	Spring	220	2 Days	1 Day	35h	P	bcrypt	184	2 ¹⁰	SR	0/1	0/6	
FSP12	1	Spring	220	2 Days		7h		bcrypt	184	2 ¹⁰	SR	1	6	
FSP13	1	Spring	220	1 Day		8h		bcrypt	184	2 ¹⁰	SR	1	6	

Table 6.3.: Evaluation of participants' initial and SecRequest-P submissions

Bold: Participants who at first delivered a plain text solution and thus, received the first security request (SecRequest-P). **Working Time** participants took to submit their initial solution. **Include SecRequest-P:** Time participants needed to add security after SecRequest-P (1 Day = 24 hours). **Active Working:** Self-reported active working time reported by participants for their final submissions. **Salt:** SR = SecureRandom, St = Static, pgC = pgCrypto, MR = Math.Random. **Copied:** Security code was most likely copied and pasted from the Internet.

20 secure initial solutions from our 43 participants. Seventeen of the 23 participants with non-secure submissions received SecRequest-P as they stored the user passwords in plain text. Most of these requests were sent to non-prompted participants (15/17). Including security prompting in the initial task description meant that there was (almost) no need to remind participants not to save passwords in plain text. For SecRequest-P, they needed on average 1.8 days (min: 15 min, max: 19 days, median: 2h 30 min, SD: 4.4 days). After SecRequest-P all participants except one at least hashed the user passwords.

Overall, 25 participants received SecRequest-G because their first or second submission achieved less than 6 points on the security scale. For SecRequest-G, participants needed on average 2 days (min: 15 min, max: 19 days, median: 1 day, SD: 3.7 days). We provide a deeper analysis of submissions after SecRequest-G in Section 6.5.5.

Participants needed on average 5.8 days for their final submissions (min: 2h 20 min, max: 32 days, median: 3 days, SD: 7.9 days). In contrast to the previous work [132], we wanted to have a more accurate time specification. Thus, we asked participants in the survey how much time they actually needed to finish the task. On average, they stated that it took them 14 hours and 30 minutes to complete the task (min: 1 h, max: 72 h, median: 10 h, SD: 14 h 50 min).

6.5.2. Prompting effect (H-P1)

As done with all the previous studies on password storage of Naiakshina et al. [130, 131, 132], we examined the effect of prompting for security in the task. We also found a significant effect of prompting on the security of participants' submissions (FET: $p = 0.006^*$, $cor - p = 0.01^*$, OR = 6.51, CI = [1.51, 33.18], family = 2). The majority of non-prompted participants submitted a non-secure solution (16 of 21); only 5 participants considered security. Of the 22 prompted participants, 15 at least hashed the passwords.

6.5.3. Java and Password Storage Experience (H-G1, H-G2)

Similar to the previous freelancer study, we did not find any effect of Java experience on the security score of the submissions (Kruskal-Wallis: $\chi^2(12) = 8.46$, $p = 0.75$, $cor - p = 0.75$). Further, we only examined submissions which did include some security, but did not find any effect of Java experience on the security score either (group: secure = 1; Kruskal-Wallis: $\chi^2(7) = 3.83$ $p = 0.80$).

In addition to that, we asked our participants whether they had stored user passwords in a database before. Only 3 of 43 participants said that they had never stored passwords before. As in the previous studies [131, 132], we did not find any significant effect on their decision to store the passwords securely in our study (FET: $p = 0.59$, $cor - p = 0.59$, OR = 0.42, CI = [0.01, 8.63]).

6.5.4. Framework (H-F1)

Similar to the previous student study [131], we did not find that the framework had a significant effect on the security score when participants stored the passwords securely (group: secure = 1; Wilcoxon Rank sum: $W = 32.5$, $p = 0.16$, family = 2, $cor - p = 0.32$). The mean score for the JSF group was 4.18 (group: secure = 1; min: 1, max: 6, median: 5, SD: 2.23). The Spring group received a higher mean of 5.33 (group: secure = 1, min: 2, max: 6, median: 6, SD: 1.41). In both studies the spring group achieved slightly higher scores, but since neither effects were statistically significant, future studies will have to decide whether the small improvement the spring framework might bring is worth conducting a study with more power.

Further, we investigated the reported usability of both APIs. We calculated the API usability scores suggested by Acar et al. [13] for the Spring group (min: 62.5, max: 92.5, mean: 74.17, median: 72.5, SD: 9.36) and the JSF group (min: 50, max: 100, mean: 68.75, median: 70, SD: 13.11). The score could range from 0-100 with 100 being the highest usability score that can be achieved. Thus, the Spring group achieved higher usability scores, however the difference was not statistically significant. (Wilcoxon Rank Sum: $W = 160$, $p = 0.08$).

Moreover, we tested for a correlation between API usability and the achieved security scores but did not find any significant correlation (Pearson: $r = 0.08$, $p = 0.57$).

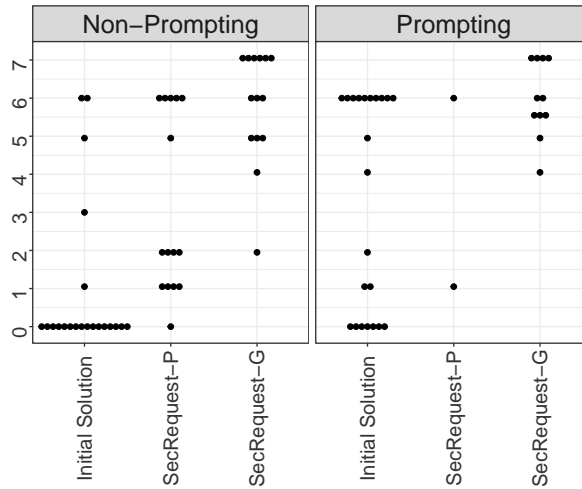


Figure 6.1.: Distribution of scores before and after the security requests for the prompted and non-prompted group.

Each point stands for one participant.

	Company Freelancer [132]				Study Freelancer (only JSF)				Study Freelancer (JSF and Spring)				
	Non-secure	Secure	Score	Total	Non-secure	Secure	Score	Total	Non-secure	Secure	Score	Total	
Initial submission	Prompting	8	13	$\mu = 2.19$ ($\sigma = 2.52$) min = 0, max = 6	21	5	6	$\mu = 2.27$ ($\sigma = 2.8$) min = 0, max = 6	11	7	15	$\mu = 3.32$ ($\sigma = 2.8$) min = 0, max = 6	22
	Non-Prompting	17	4	$\mu = 0.86$ ($\sigma = 1.96$) min = 0, max = 6	21	6	5	$\mu = 1.91$ ($\sigma = 2.59$) min = 0, max = 6	11	16	5	$\mu = 1$ ($\sigma = 2.07$) min = 0, max = 6	21
	Total	25	17	$\mu = 1.52$ ($\sigma = 2.33$)	42	11	11	$\mu = 2.09$ ($\sigma = 2.64$)	22	23	20	$\mu = 2.19$ ($\sigma = 2.68$)	43
After SecRequest-P	Prompting	2	1	$\mu = 2$ ($\sigma = 3.46$) min = 0, max = 6	3	0	1	$\mu = 1$ ($\sigma = 0$) min = 1, max = 1	1	0	2	$\mu = 3.5$ ($\sigma = 3.54$) min = 1, max = 6	2
	Non-Prompting	4	10	$\mu = 2$ ($\sigma = 2.29$) min = 0, max = 6	14	0	6	$\mu = 2.17$ ($\sigma = 1.94$) min = 1, max = 6	6	1	14	$\mu = 3.13$ ($\sigma = 2.36$) min = 0, max = 6	15
	Total	6	11	$\mu = 2$ ($\sigma = 2.33$)	17	0	7	$\mu = 2$ ($\sigma = 1.83$)	7	1	16	$\mu = 3.18$ ($\sigma = 2.31$)	17

Table 6.4.: Comparison of (non-)secure solutions and the security score

In this study, we asked participants to evaluate the API usability after possible security requests, so we can be certain that all the participants used security mechanisms within the frameworks before giving us their assessment. In the student study [131] however, there were no follow up security requests and thus there were participants who did not use any security mechanisms. Since their experience with a framework was reported based on functionality aspects only, we do not draw a direct comparison to our study.

6.5.5. Security Guidelines (NIST and OWASP)

Figure 6.1 shows the distribution of scores for the prompted and non-prompted groups for initial submissions and after participants received SecRequest-P and/or SecRequest-G. The figure visualizes how the distribution evolved after each request. The achieved scores rose after each request. Out of all 43 participants, 25 received SecRequest-G with web links to NIST and OWASP.

The evaluation of these submissions is available in the Appendix (Table D.1). The mean security score achieved was 5.86 (min: 2, max: 7, median: 6). Ten of the 25 participants used Argon2 to store the passwords and thus, achieved 7 points on the security score. Five of the 25 participants used bcrypt and achieved 6 points. The remaining 10 of 25 participants submitted solutions below 6 points, like PBKDF2 with insufficient parameters, SHA-1, or MD5.

In the follow-up survey—independently whether participants received SecRequest-G—we asked participants whether they know about and have experience with NIST or OWASP sources for user password storage. 58% (25 of 43) of all participants reported that they had heard of NIST or OWASP before. 47% (20 of 43) participants stated to have followed one (or both) of the guidelines in their submissions (including participants without SecRequest-G). Of the 25 participants who received SecRequest-G, 15 reported to have heard of NIST or OWASP. Consequently, 10 participants received instructions from us with NIST and OWASP sources, but indicated to not know the sources.

When comparing both guidelines, NIST offers a more theoretical and complex recommendation, while OWASP offers Argon2 example code in Java with clearer recommendations. We investigated whether our participants had copied and pasted code from the OWASP guideline and found that out of 10 participants who implemented Argon2 after our second security request, 8 copied and pasted the code from the OWASP guideline. We found the same comments in the programming code as on the website.

In the following we present an analysis of participants' experience with the guidelines based on their open-question answers in the survey and the chat communication.

Guideline Experience with SecRequest-G. Our participants mentioned that reading the guidelines has helped them to increase their knowledge. Participants found the guidelines useful, as they provided them with details, they were not aware of:

"I was unaware of Argon2 and the weakness in PBKDF2-with-HMAC and hadn't thought of a few things that were mentioned in the guidelines (max password length for DoS protection, Unicode normalization)" (FJP3).

FSN5 reported to like OWASP because it instructed him to use existing hashing and salting algorithms instead of implementing them himself:

"One of the OSWAP design principles is to keep security simple. In the registration process I avoided implementing own salt and hashing algorithms [...]. This reduces chances of making security mistakes."

Guideline Experience without SecRequest-G. Ten of 43 participants (21%), who did not receive SecRequest-G, reported to have heard of NIST or OWASP before. Five of them (FJN4, FJP4, FSN2, FSN12, FSP5) stated that they did not follow the guidelines of the organizations in our task. As reasons FJP4 noted that he heard of the organizations, but never implemented their guidelines before and FSP5 stated: *"I could develop it without these practices."* Both used bcrypt in their initial solution and received 6 points in the security score. FSN11, FSP1, FSP3, FSP10, and FSP13 reported to have followed one of the guidelines. Four of them used bcrypt in their first submission and FSP10 used bcrypt in his second submission after SecRequest-P. FSP1 reported about NIST:

"NIST guidelines were easy to follow as they are in line with security best practices."

6.5.6. Sample Comparison

In this section we present a direct comparison between the previous freelancer study [132] and our replication study. In the following we denote this study as *Study Freelancer* and the original study as *Company Freelancer*, since a company deception was used as study design. Unlike in the *Company Freelancer* study, where only JSF was used as framework, this study also looked at Spring as a between groups condition. However, for the examination of the deception effect, we restrict our comparison to the JSF participants of our study, since only there a direct comparison can be made. Figure 6.2 displays a distribution of participants' initial submission security scores from the both studies *Study Freelancer* and *Company Freelancer*. For our study, Spring and JSF results are reported separately. Considering the prompted and non-prompted groups, the distributions of the obtained scores are fairly similar.

Table 6.4 summarizes participants' initial and SecRequest-P related security results of both the *Study Freelancer* and *Company Freelancer*. Since in the original study SecRequest-G was not introduced to participants, we do not consider it for the comparison of the both studies. Table 6.4 shows the count of participants from the two groups (prompted and non-prompted) and how many of the solutions were secure or non-secure. The proportions of participants in each group in both freelancer samples appear to be similar. Further, the mean score values from the company freelancers and study freelancers (only the JSF group) are similar as well.

We did not find any significant difference between the JSF security scores of the Study Scenario ($\mu = 2.09, \sigma = 2.64$) and Company Freelancer Scenario ($\mu = 1.52, \sigma = 2.33$) in the scores of the initial submissions (group: secure = 1 & group = JSF; Wilcoxon Rank sum: $W = 86, p = 0.73, r = 0.09$). However, a lack of a statistically significant finding does not mean there is none. With a large enough sample even small differences will result in a statistically significant finding. Power analysis is necessary to avoid Type II static errors (i.e., false negatives). In this study, power analysis would have established whether the lack of a statistically significant difference between the replicated and original results was meaningful, or if it was just an artifact of too few participants. Future studies will have to decide whether the differences shown above are worth re-examining with a larger sample size.

Implementation Time

We compared the implementation time to submit the initial solution of our study with the prior freelancer study [132] (min: 1 day, max: 8 days, mean: 3 days, median: 3 days, SD: 1.88). We did not find any significant difference between the implementation time of both JSF groups (Wilcoxon Rank sum: $W = 457, p = 0.83$). Further, we compared the time spent on the first security request. In [132], the participants needed on average 6.4 hours (sd 7.3 h, median 3.17 h). We did not find a significant effect in both freelancer studies either ($W = 74, p = 0.52$). The same caveats about the lack of power apply as above.

Study Announcement: Self-reflection

In the follow-up survey our participants were asked whether they would have stored the password securely if it had not been a study. Interestingly the answers split in half, 22 reported to would

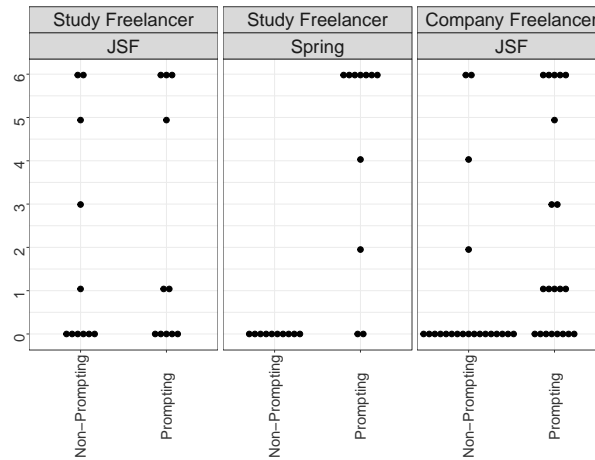


Figure 6.2.: Initial submission score comparison of the previous freelancer study [132] (Company Freelancer) and our replication study (Study Freelancer).

Each point stands for one participant. In the Company Freelancer JSF group, Naiakshina et al. had almost twice as much participants as in the JSF group of this study, because we also investigated Spring as another value for the framework variable.

have stored the passwords securely, and 21 reported that they did not think that they would have stored them securely. Six of the 22 who reported that they would have performed differently if it was not a study, stored the passwords in a sufficient way in the initial submissions. The results of the freelancer study of Naiakshina et al. [132] where the participants were not aware that they were participating in a study until they finished programming, shows that participants in developer studies can overestimate their own security awareness. We found that the reported details do not always fit the actual code submission.

IV	DV	Statistical Test	Company Freelancer [132]	Study Freelancer
Prompting	Secure	FET	$p = 0.01^*$ OR = 6.55, CI = [1.44, 37.04]	$p = 0.006^*$ OR = 6.51, CI = [1.51, 33.18]
Java Experience	Score	Kruskal-Wallis	$p = 0.21, r = 0.12$	$p = 0.75, r = 0.04$
Stored Passwords Before	Secure	FET	$p = 0.17$ OR = 0, CI = [0,3.87]	$p = 0.59$ OR = 0.41, CI = [0.04, 2.69]
Framework	Score	Wilcoxon Rank sum	- -	$p = 0.16$ group: secure = 1, $r = 0.3$

Table 6.5.: Summary of all tests across the different samples

IV: Independent variable, **DV:** Dependent variable, **Company Freelancer:** Freelancers with study deception [132], **Study Freelancer:** Freelancers without study deception, **OR:** Odds ratio, **CI:** Confidence interval. The IV framework was not examined for freelancers in [132]. Significant tests are marked with *.

6.6. Discussion

Methodological implications for developer studies: Our results suggest that freelancers are a useful sample for usable security developer studies. Response rates are higher than on GitHub. The studies can be conducted online and it is possible to reach developers from all over the world. Freelancers could provide more experience with real world projects and a wide range of age and experience. Since freelancers on the platform most likely do not know each other, the probability of participants communicating with each other about the study task and solutions is lower than for example, in a CS student sample. Additionally, freelancers are a convenient sample as they are looking for jobs and work with their own devices.

In comparison to the field study of Naiakshina et al. [132], where the study itself was concealed, our study was conducted by openly communicating the study context to the freelancers. Using the same sample size and same study protocol (minus the study deception), we got similar results as Naiakshina et al. We got similar effect sizes, directions, and statistically significant results for the same tests as they did and did not get any that they did not. In Table 6.5 a comparison of both studies and tests can be found. In both studies prompting lead to more security. The impact was significant in both samples, so we observed the same treatment effect (Company Freelancer: OR = 6.55 and Study Freelancer: OR = 6.51).

Neither study found a significant effect of previous password storage experience. However, in both cases only very few participants reported to have no password storage experience at all (Company Freelancer: $n = 2$ and Study Freelancer: $n = 3$). Therefore, the results should not be over-interpreted. Furthermore, Naiakshina et al.[132] did not find an effect of Java experience on security. In this study, we did not find an effect either. However, the direction of the correlation is the same in both studies. Future studies will have to decide whether to examine this effect with larger sample sizes.

In contrast to the previous freelancer study, we also tested the two frameworks Spring and JSF as done in the student study [131]. In [131], Naiakshina et al. did not find any significant difference between Spring and JSF. We did not find a significant difference between the two frameworks either. In both studies the mean security score was higher for the Spring framework but future studies with more participants would be needed to examine this effect further.

We conclude that for this study the removal of the deception element does not seem to have changed any outcomes and all relevant results gained from the original study with deception have also been gathered by this one without. Since deception should only be used when necessary, for this study we would not recommend to use it again in this specific context. While these results certainly do not generalize to all developer security studies, it is an important first indication that freelance developers recruited as part of a study behave similarly to when they are hired for a regular job. Therefore, our findings also offer an early indication that platforms such as Freelancer.com may be promising platforms for developer recruitment to supplement other channels such as CS students and GitHub developers.

Security guidelines: Acar et al. [11] showed in their programming experiment that using standard documentation without access to other sources such as the Internet lead to more secure code. However, the set-up of the experiment was rather artificial. In the real world developers use the

Internet to find solutions while programming [131]. The standards are available but only a few developers use them or are even aware of them.

We found that concrete security policies with web links to the guidelines did increase the score of the password storage code. Ten participants were able to achieve full 7 points although no participants in the past studies and in the initial submissions achieved such a level for security. Even though some participants were able to use secure industry standards without being requested with the specific policies, a number of participants reported to know the guidelines. However, these participants were only able to score at most 6 points. This finding emphasizes the need for explicit encouragement of using security policies. The more support the policies offer the more secure the code can be.

Performance-based payment using milestones: Milestone payment is the recommended payment method on Freelancer.com. We split the payment into three milestones and chose the milestones for the initial submission and the (security) code review in a 1:1 ratio. It has to be investigated whether different ratios might result in different security results. Different performance rewards could help to increase security and the security awareness as well. We chose this ratio as we did not know how many security requests a participant would need as this depends on the password storage security in the initial and follow-up submission. In this study we chose the security to weight as much as the initial functional solution. We note that rewarding security performance as a study variable might lead to better initial solutions. In future research, this has to be evaluated with other scenarios and developer studies.

6.7. Summary

One major issue of usable security developer studies is their ecological validity. In the password-storage study of Naiakshina et al. [130, 131], CS students claimed that they would have behaved differently if they would have worked for a company. In a follow-up study, Naiakshina et al. [132] concealed the study context and hired freelancers for the same project. This form of deception requires additional study design work to maintain the deception. Frequent use of deception can cause problems as well. Both these issues make the use of deception for this kind of developer study something one would want to avoid if possible. Therefore, we replicated the deception study of Naiakshina et al. [132] without deception and compared the results. Overall the results were very similar leading us to propose the following recommendations:

- **When trying to get ecologically valid results for freelance developers, deception is not always necessary.** In our example running the study openly produced very similar outcomes compared to hiring freelancers for real. We got similar effect sizes and directions for the same tests as Naiakshina et al. in [132]. However, our study presents only one data point and further research is needed on the use of deception in other studies covering other security issues, tasks and scenarios, as well as with other types of developers.
 - **Instruct developers to use security-guidelines.** We found that providing participants with specific guidelines for password storage can increase the security of their solutions drastically. Almost all our participants were able to implement secure password storage after being provided with specific security guidelines. If guidelines offer code examples, they are more
-

likely to be implemented and included into the developers' code. Thus, we recommend designers of security guidelines to give specific code examples of secure code. We further recommend organizations to provide developers with specific security guidelines to receive software with state-of-the-art security standards. If guidelines might not be known to the employer, we recommend to include at least security prompting in the task to raise security awareness.

7. Task Design: Comparison of Code-Reviewing and Code-Writing in Developer Studies

Disclaimer: The contents of this chapter were published as parts of the publication "Code Reviewing as Methodology for Online Security Studies with Developers - A Case Study with Freelancers on Password Storage", presented at the 17th USENIX Symposium on Usable Privacy and Security (SOUPS 2021) [61]. The paper was published in cooperation with my co-authors Alena Naiakshina, Anna Rasgauski and Matthew Smith. I led the conception of the study. Alena Naiakshina and Matthew Smith took part in the conception. The study design was conducted by Anna Rasgauski. Supported by Matthew Smith, I created and finalized the study design. Anna Rasgauski conducted the study under my supervision. The participants' open answers were coded the survey and analyzed by Anna Rasgauski and me. I conducted statistical data analysis. I led the write-up of the paper in cooperation by Alena Naiakshina, Matthew Smith and Anna Rasgauski.

7.1. Motivation

Code reviewing is a technique applied at the end of the Software Development Life Cycle (SDLC), used as one of the final steps by software developers to ensure programming code quality before software release. Thus, software developers change their perspective from a code creator to a code inspector, which might affect their security awareness. While knowledge on code reviewing in the field of software engineering exists [33, 113, 164], only little is known about this methodology within a security context [70]. Acar et al. [12] called for more studies on developers' security behavior and on the study methodology of security developers. While most of the previous work within this context includes surveys (e.g., [59, 25, 117, 174, 138]), interviews (e.g., [59, 117, 130, 174, 183, 100, 24, 172]) or programming tasks (e.g., [131, 131, 132, 133, 114, 192, 11, 13, 186, 185, 136, 191, 135, 154, 94]), we explored code reviewing as a promising methodology for developer security studies.

Conducting security studies with software developers can be difficult due to recruitment and study compensation challenges [12, 117, 161, 11, 14, 114, 192, 45, 63, 133, 131]. Programming tasks can also often take more time than professionals can afford. Naiakshina et al. [130, 131, 132, 133] conducted a number of studies where computer science (CS) students, freelance developers and software developers from companies were required to complete the registration functionality in a web application. The authors investigated participants' security behavior with a focus on the storage of user passwords in a database. According to the study design and participants' feedback, the study lasted around 8 hours. Recruiting a high number of employed developers who had time outside of their normal working hours for a one-day study was reported to be extremely difficult. Thus, their sample size is not as large as they would have wished.

Therefore, researchers often tend to design programming tasks in such a way that software developers only need to solve small and short tasks (e.g., [13, 11, 14, 168]). For example, Acar et al. [14] conducted a security developer study with GitHub users and provided them with programming tasks which were "short enough so that the uncompensated participants would be likely to complete them before losing interest, but still complex enough to be interesting and allow for some mistakes." While Acar et al. did not mention security or privacy in the recruitment message at all, Naiakshina et al. tested if explicitly asking participants for secure password storage (security prompting) would affect their solutions. They found that security prompting had a

significant effect on participants solutions. Additionally, participants tended to concentrate on the functionality of the software first, before working on security aspects [130, 132, 133].

To provide deeper insights into this research field, we tested code reviewing as a promising methodology for security studies with developers. Instead of asking developers to program a piece of code, we showed them functional code snippets and asked them to write code reviews about the snippets. We based our code snippets on the participants' submissions from the previous freelancer study from Naiakshina et al. [132] and also recruited freelance developers. This allowed us to compare the results of the code review task with the findings of the previous study containing a programming task. While we offer insights on freelancers' behavior in code reviewing tasks on a primary level, we also discuss code reviewing as a methodology for developer security studies on a meta-level. Our main research questions are as follows:

RQ1: How do developers behave when reviewing code in a security-critical task such as password storage? We were interested to find which criteria the developers mostly base their reviews on in security-critical code, and whether they would be able to indicate the security issue, even when being presented with distraction tasks not related to security. Additionally, how detailed would their security problem description and suggestions for improvement be? Would they suggest to release the code even though it contains security-critical issues?

RQ2: Which factors have an influence on developers' security awareness? In particular, we investigated whether prompting, programming experience or code snippets with different password-storage issues (plain text, Base64, MD5) have an influence on whether developers find the security issues.

RQ3: How much time do developers dedicate to security and do they feel responsible for security in a code review? We asked the freelance developers to indicate how much time they spend with security in code reviews and whether they feel responsible for security. These results were compared to their self-reported responsibility and time spent on security in programming tasks.

RQ4: Comparing the results of a programming and a code reviewing task on password storage, which methodological implications can we conclude? To provide insights for security studies with developers, we discuss the advantages and disadvantages of code reviewing as a study methodology compared to programming tasks.

7.2. Methodology

Past work showed that software developers perceive security as more of a secondary task during programming and thus need to be explicitly asked to consider security aspects in the task requirements [130, 132, 133, 136]. It is however unclear, whether this holds true for code reviewing. The fact that code reviews are usually written at the end of the SDLC might affect developers' security awareness and thus improve the quality and security of the code.

In this work, we investigated whether software developers think of security when writing a review for security critical code, such as user password storage. For this we set up an online survey, for which we recruited freelance developers on Fiverr.com [4]. The complete survey can be found in the Appendix E.1. Half the participants were prompted for password-storage security prior to writing the review, and the other half were asked without being prompted to write the

review. Hence, we explored the independent variable (IV) *security prompting* with the two values prompting and non-prompting. Additionally, we investigated whether different password storage implementations affect the code reviews. Thus, the participants were shown at random one of three insecure code snippets (plain text, Base64, MD5). Hence, our second IV variable was the *code snippet* each participant received, leaving us with a total of six conditions within our study. We conducted a between-subjects study, where each participant randomly received one of the three snippets.

Apart from prompting the prompted groups of participants to ensure the password is stored securely, we did not give any criteria to complete the review. We wanted to find out which criteria the participants chose and which issues they found without further requests. We recorded all questions received from the participants during the task and recorded our answers to these in a play-book to avoid giving more information to some participants than to others (see Appendix E.2). In addition to that, we provide further insights on a meta-level for using code reviewing as a new, promising methodology for developer studies within a security context.

We conducted a pilot study with one participant to test the survey and to get a better time estimation for the task. The participant finished within two hours. After correcting minor issues with the survey, we conducted the actual study between May and July 2020. We asked our participants to complete the code reviewing task within one week. Thus, we hoped to increase the number of participants.

7.2.1. Survey

In the survey, we showed the participants one of three code snippets, each of which contained a password storage implementation. The participants were asked to write code reviews for the snippets. We also asked them to list criteria on which they based their reviews and if they would release the code as it is, with minor adjustments or not at all. After the participants finished their code review, they were requested to explain the concepts of hashing and salting. Further, we asked them about their code reviewing experience and how much of a priority security is to them.

We switched off the back button to prevent the participants from changing their code reviews after being asked for code security. That way we aimed to get an unbiased view on the code reviews and to avoid priming participants for security by the survey.

Since we did not ask participants to write but only review programming code, we included a small programming test to the survey, which we also used in [59]. We recruited software developers for an online survey study on security warnings. To assure that our participants really had programming skills, we designed a multiple choice question with a code snippet where “hello world” was printed out backwards. About 74% of the participants recruited online on the survey platform Qualtrics failed the test, although all of them indicated to have programming skills. To ensure data quality in this study too, our participants were also shown this multiple choice question. Finally, the participants had to answer demographic questions.

Table 7.1.: Demographics of participants (n = 44)

Age	min: 19, max: 35	sd: 3.83	median: 24.0	mean: 25.06
General Programming Experience	min: 1, max: 12	sd: 2.43	median: 4.0	mean: 4.46
Java Experience	min: 0.5, max: 10	sd: 2.3	median: 2	mean: 3.19
Gender	Female: 3	Male: 40	Prefer to self-describe: 1	
Occupation	Freelance Developer: 27 Academic Researcher: 1	Industry Developer: 8 Industry Tester: 1	Undergraduate: 2 Graduate: 1	Other: 3 Freelance Tester: 1
Country of Residence	Pakistan: 21, India: 8 Nigeria: 1, Turkey: 1	UK: 3, Portugal: 1 Malaysia: 1, Italy: 1, US: 1	Burkina Faso: 1 Bangladesh: 1	Morocco: 1 Sri Lanka: 2, NA: 1

7.2.2. Code Snippets

For the code review task, we chose code snippets submitted by freelancers from the study conducted by Naiakshina et al. [132] for two reasons. Firstly, it was programming code created by freelance developers who believed they were working on the registration functionality for a real company which was to be submitted for release. Secondly, it allowed us to compare our code review results with the findings of the programming code analysis from Naiakshina et al. with regards to both the effectiveness of security prompting and the accuracy of the submissions.

Additionally, we wanted to test whether different programming code snippets have an influence on our participants' submissions. We concentrated on the bad practices used by freelancers in [132]. While we decided upon a plain text code snippet as a baseline, we also added one snippet using MD5 as a hashing function and another using Base64 encoding, as these were prevalent within the freelancers' submissions in [132]. We made sure that the three snippets had an approximately similar length (120-130 lines). To further improve their comparability, we adjusted the selected snippets in the following way. First, to reduce the risk of comments influencing the code review, we deleted all comments from the chosen snippets. To look more realistic, but still stay comparable, we added generic comments which were the same for all snippets. We added comments to the head of the class as well as in the main functions, but not to the trivial setters and getters. Second, we rearranged the order of the functions so that all participants would see the functions in the same order when reviewing the code, as we did not want the function order to influence our results. Third, we added two distraction tasks to all of the snippets:

- 1) *Exception swallowing*: An empty catch block is considered to be bad practice as possible exceptions would be ignored.
- 2) *Logical mistake*: Within an if-loop we used only one "=" in the condition. The condition is therefore always true since an assignment is executed instead. We expected that since the participants are eager to find mistakes in a study on code reviewing, these distraction tasks could divert the attention of the participants from the password storage implementation. The three code snippets can be found in the Appendix E.3.

7.2.3. Participants

Like Naiakshina et al. [132], we wanted to recruit freelance developers on Freelancer.com. However, our project was repeatedly denied by the platform with generic explanations such as: "Your project shows behavior that is contrary to our Code of Conduct." We contacted the platform's

Table 7.2.: Number of participants per group (n = 44)

	Plain text	MD5	Base64
Prompted	8	7	8
Non-prompted	6	7	8

support service several times to clarify that we wanted to conduct a scientific study with freelancers. However, we were not able to solve our issues at that time and thus decided to use another freelancer platform for the recruitment of participants: Fiverr.com [4].¹

In the freelancer study by Naiakshina et al., participants received either €120 or €220 for participating in a study of six to eight-hours. No significant difference was found on the security of the submissions between the different payment groups. Since our study was estimated to take one hour and the participant in the pilot study needed two, we decided on a compensation of \$50.

We posted our project in four iterations receiving up to 15 applications per posting. With each iteration, the number of repeated offers increased. On Fiverr.com it is required to attach categories and subcategories to the post. Our post was included in the categories “Programming and Tech”, “Web-programming” and “Java.” In total we received 61 applications to take part in our study. All except four were invited to the study; reasons for not being invited to take part included being under 18 or not having any programming experience. Four freelancers did not respond upon our invitation, six did not want to participate, and two participants canceled after invitation before starting with the study. Finally, 45 freelancers completed our survey. To assure data quality, we excluded one participant from our data set who was not able to answer the “hello world” backward question from our study [59].

Table 7.1 summarizes the demographics of our participants. Out of 44 participants, 40 reported to be male, 3 female, 1 preferred to self-describe. They reported to be between 19 and 35 years old (mean: 25.06 years, median (md): 24 years, sd: 3.83). Further, most of the participants reported to live in Pakistan or India. The general programming experience ranged between one and 12 years with a median of 4 years. All except 2 had at least 2 years of general programming experience and all but 8 reported to have at least 2 years of Java experience (min: 0.5, max: 10, md: 2, mean: 3.19). Most (27) named freelancing as their main profession. All except two participants reported to have reviewed code by others in the past. Table 7.2 shows the number of valid participants in each group.

7.2.4. Evaluation

7.2.4.1. Security

We evaluated the security of participants’ code review submissions in the following way. First, we introduced a binary variable *found password storage issue* with two values: 1: participants stated in

¹Another reason the support service of Freelancer.com offered us was: “Academic cheating is not allowed.” As pointed out by Naiakshina et al. [132], it seems that students often use this platform for hiring freelancers to do their university homework. We are still in contact with the enterprise department of Freelancer.com, which reassured us that university studies are welcome to use their platform. It seems that an enterprise self-service would have solved our previous issues.

their reviews, that they found some issues with password storage security; 0: participants did not state in their reviews, that they found some issues with password storage security.

Second, to identify how accurate our participants' code reviews were with regard to the secure password-storage parameters, we used the *security score* of Naiakshina et al. in [132]. Participants received 2 points for hashing and salting the user passwords, another 2 points if the salt was randomly generated and at least 32 bits in length, and another 3 points for iterations, a memory-hard hashing function and if the hash's derived length was at least 160 bits long [130]:

1. The end-user password is salted (+1) and hashed (+1).
2. The derived length of the hash is at least 160 bits long (+1).
3. The iteration count for key stretching is at least 1 000 (+0.5) or 10 000 (+1) for PBKDF2 and at least $2^{10} = 1\,024$ for bcrypt (+1).
4. A memory-hard hashing function is used (+1).
5. The salt value is generated randomly (+1).
6. The salt is at least 32 bits in length (+1).

7.2.4.2. Qualitative

The code reviews were evaluated qualitatively with inductive content analysis [73]. We decided upon an inductive coding method, as opposed to a deductive coding method, as we did not want to assume what our results would be. The evaluation process included open coding and creating categories. Since we used the "independent parallel coding" approach of David R. Thomas [171], we compared two sets of categories and report the inter-coder agreement for them. The two sets of categories were subsequently merged into a combined set. We calculated the inter-coder agreement Cohen's Kappa (κ) and received an agreement of 0.82. Fleiss et al. considered a value above 0.75 a good level of coding agreement [87].

7.2.4.3. Quantitative

We additionally conducted an exploratory quantitative analysis. We established the variable from the reviews (found password storage issue) and evaluated the effect of our two IVs (security prompting, code snippet) on this variable using Fisher's exact tests (FET) [80, p. 816]. To test for correlations, we used the Pearson's correlation coefficient. To examine effects in continuous data, we used Wilcoxon Rank sum tests. All tests referring to the same dependent variable (found password storage issue) were corrected using the Bonferroni-Holm correction. The corrected p-values are referred to as *cor - p*.

7.2.5. Ethics

The institutional review board of our university reviewed and approved our project. We provided the participants of our study with a consent form outlining the scope of the study, the data use and retention policies; we also complied with the General Data Protection Regulation (GDPR). The participants were informed of the practices used to process and store their data and that they could

withdraw their data during or after the study without any consequences. Also, the participants were asked to download the consent form for their own use and information.

7.2.6. Limitations

The participants who took part in this study were recruited on Fiverr.com. They may not be representative for all developers and results may even differ among different freelance platforms. Code reviews are usually performed in companies, so the freelancers might not have had experience with code reviews. However, we aimed to test code reviewing as a study methodology for developer studies as opposed to studying the code review experience.

The majority of participants were non-native English speakers and their responses were not always so clear to understand. Some participants may have had trouble understanding the questions. While this is not desirable for a study, it still represents a realistic scenario, since freelancers with the same issues are hired for real life projects. Moreover, all participants were informed that this project is part of a study. It could be that the participants would have behaved differently had they been writing a code review for a real life project.

We conducted an a priori power analysis with an effect size from Naiakshina et al. from [133] to calculate the necessary sample size to prevent type II errors of falsely rejecting null hypotheses. The required sample size turned out to be 45 persons per group. However, we were not able to recruit enough freelancers on Fiverr.com, even though we used multiple rounds of recruitment and posted the project repeatedly on the platform. The recruitment of software developers is a challenging task and small sample sizes can limit the method's potential and the generalizability of results. Therefore, our analysis needs to be considered as an explanatory first glance on the problem.

7.3. Results

The results section is structured as following. First, we present the findings of our qualitative analysis. Second, we report the results of our exploratory quantitative analysis. Third, we compare our results with a similar study containing a programming task on password-storage. We report statements of specific participants by labeling them according to their conditions. The first letter of the label refers to Prompting or Non-Prompting. The second denotes the code snippet used (Base64, MD5, or Plain text). While qualitative analysis is more frequently used to explore phenomena, we still provide the numbers of participants to give an indication of the frequency and distribution of themes. An overview of the evaluation of participants' submissions can be found in the Appendix E.4.

7.3.1. Qualitative Analysis

The reviews of our participants differed in quality, word count and content. The majority of participants (36 of 44) reported to have reviewed the snippet manually, 4 said that they used an IDE (Eclipse, NetBeans, Visual Studio Code) to check the code, and 4 reported to have used a static analyzer (PMD, sonarlint, findbugs, codacy). With such a small sample size, we could not draw conclusions but it might be worth exploring the use of static analyzers in future studies. On average participants needed a median of 83 minutes to complete the survey. The fastest was submitted after

12 minutes. Some participants took more time since the deadline to complete the project was set to one week.

7.3.1.1. Participants' Review Criteria

To provide insights into the security awareness and focus of freelancers, we evaluated the criteria which participants mentioned to have looked for in their code reviews. A detailed list of criteria mentioned by our participants is available in the Appendix E.5. We categorized the answers as follows:

Implementation: A total of six participants said that functionality was one criteria they looked for. Logic was mentioned by eight participants to be an important topic to look for in source code. One participant reported to have looked for maintainability and two mentioned performance and efficiency (PB5, NP4). A large number of participants (14/44) reported to also have checked for error handling.

Testing and bugs: NM3 said that quality assurance was one criteria to check for, while PB7 checked for unit tests and whether all scenarios are considered. NM1, NM5, and NP4 said they looked for bugs or errors in the code. A number of participants mentioned syntax to be a criterion to look for.

Standards and validation: NB1 and PB1 said that they checked whether code standards are met in the code. Three participants mentioned that code format was a criterion they checked for (PB2, NM4, PP5). Nine participants reported that they looked for the correct usage of get and set methods. NM1, NP2, NB8 mentioned the inspection of the model view controller architecture. Further, several participants reported to check imported packages and libraries. Some participants mentioned input validation and null checks (PM6) as criteria they checked the code for. Additionally, several freelancers reported to have checked for code style issues; e.g., camel case or naming conventions in the code. PB1 and PB2 wrote that they looked for duplicated code or unused code. Furthermore, four participants reported to assess the code complexity. Some participants said that they checked the code for readability and comments.

Security: Security in general was mentioned as a criterion by 10 participants. Eleven participants specifically included password storage security in their criteria. Data security was mentioned by 4 participants.

7.3.1.2. Found Password Storage Issue

Thirteen participants specifically mentioned in their reviews that secure password storage is an issue. Of these, only 2 were non-prompted. To solve the issue, PP8 suggested to use an external authentication service:

“Depending on the application it may also be better in this case to simply use an external auth service such as that offered by google” (PP8).

Some prompted participants misinterpreted our prompting task description “Please ensure that the user password is stored securely” as password validation (NM3, PP4, PB8). For example, PB8 included secure password policies in the review but failed to detect the insecure password storage method:

“The password must be at least 8 characters long. The password must have at least one uppercase and one lowercase letter. The password must have at least one digit. This needs to be updated” (PB8).

NM3 mentioned another password validation policy issue:

“The most important part is the one you are not verifying the password what if it is equal to username. You should know that any person who is trying hit and try on the passwords will definitely first try to enter same username and password and he might be successful in your code and it’s the worst part in security risks.”

We also found that a number of participants used “password encryption” as a suggestion in their review, which is a concept not recommended for secure user password storage in a database. Furthermore, the prompted participant PB6 wrongly stated that the code snippets contained SQL and JAR injections, but did not mention insecure password storage as an issue. Finally, PM3 perceived the password storage implementation as “too complex” and asked in their review for more comments in the code snippet. A detailed list of code issues reported by our participants is available in the Appendix E.6.

7.3.1.3. Security Score

A number of participants commented on the password storage methods of the code snippets. Some participants explicitly said that the password storage functions were sufficient (NM4, NM5, PB8), others saw issues with the functions.

For example, a number of participants recommended using secure hashing functions like bcrypt (PM1, PM5), PBKDF2 (PM6), ARGON2 (PM1) or scrypt (PM5). In contrast, others recommended less secure functions such as MD5 (PP4), SHA-1 (PM4, PP4) or SHA-2 (PM1) (without mentioning iterations). It is noted that SHA-1 and SHA-2 can be secure when used with a key derivation function like PBKDF2, but we cannot assume that the participants mean this if they do not include it in their reviews. The code reviews of the 13 participants who identified user password storage security as an issue often lacked details. Thus, we only were able to grade 11 participants by using the security scale. Table 7.3 summarizes all the participants who found an issue with user password storage and their score according to the security scale. Only two participants specifically mentioned that a salt should be used.

7.3.1.4. Distraction Tasks

Some participants found the issues we introduced as distraction tasks. The logical mistake was found by 8 participants and the exception swallowing was mentioned by 12 participants.

NM5 falsely stated that error handling was done correctly. Out of all the 16 participants who found at least one distraction issue, 13 did not find the password storage issue, and 3 did (PM7, PB1, PM4). This might indicate that the distraction issues could have indeed distracted the participants from security. However, from the 13 participants who found the password storage issue, only 3 found at least another distraction issue.

7.3.1.5. Ready for Release?

We asked our participants to choose whether the code can 1) be released, 2) be released but the issues mentioned in the review need to be fixed for the next update, or 3) whether the code did not pass the review. Out of 44 participants, 2 said that the code can be released. Both found no security issues. Another 17 said that the code can be released but the issues should be fixed for the next update. 12 of these 17 participants did not find the security issue, so they referred to non-security related issues which needed to be fixed. The remaining 25 said the code did not pass the review and should not be released until the issues were fixed. 17 of these 25 participants, however, did not find the security issue with password-storage, so they have based their decision to not release the code on issues other than security. A detailed overview can be found in the Appendix E.4.

7.3.1.6. Participants' Definition of Hashing and Salting

After the review completion, we asked our participants to give the definition of hashing and salting passwords. We wanted to find out how many participants were able to correctly define hashing and salting of passwords. On average the participants took 8 minutes to answer this question (md: 5). We evaluated the responses and also tracked whether the participants left the tab inactive while answering the question. Out of 44 participants, 20 participants left the tab inactive for some time while 24 did not leave the tab to answer the question. The participants who left the tab inactive spent a median of 3 minutes outside the tab (mean: 6 minutes). This might indicate, that participants were searching the Web for the answer.

The majority of participants, 63% (28 of 44), gave a correct definition of hashing and salting for password storage. We checked whether the responses matched with definitions from the Internet indicating they were copied and pasted. We found that 8 participants copied the entire definition or parts of their definition from the Internet.

7.3.1.7. Security Responsibility

We asked our participants whether they felt responsible for end-users' security when writing or reviewing code. Figure 7.1 visualizes the responses. The "disagree" options 1,2,3 are summarized on the left. By contrast, the "agree" options 5,6,7 are summarized on the right. The percentages next to the bars are the sums of participants' ratings for each side. Neutral (4) is counted as an own point. For both tasks, the participants reported to strongly agree with the statements. However, after completing their reviews, we asked the participants whether they had ensured that the user password was stored securely. Out of 44 participants, 29 answered that they ensured that they had, which did not correspond to the evaluation of their reviews. In fact, only 13 participants correctly mentioned that insecure password storage was an issue in the snippets. This suggests a social

Table 7.3.: Security score of participants who found the password storage issue

	Snippet	P	Score	Salt	Suggestion
NP1	Plaintext	n	0	-	“hashcode generation or convert in Hexa or other formats”
NP5	Plaintext	n	0	-	-
PB1	Base64	p	0	-	“an encoded format”
PB2	Base64	p	2	Y	-
PM1	MD5	p	3	-	“SHA2, Argon2, bcrypt”
PM4	MD5	p	1	-	“SHA-1”
PM5	MD5	p	3	-	“bcrypt, scrypt”
PM6	MD5	p	4	Y	“PBKDF2”
PM7	MD5	p	0	-	-
PP1	Plaintext	p	1	-	-
PP2	Plaintext	p	0	-	-
PP3	Plaintext	p	1	-	-
PP4	Plaintext	p	1	-	“SHA, MD5”

p = Prompted, n = Non-prompted

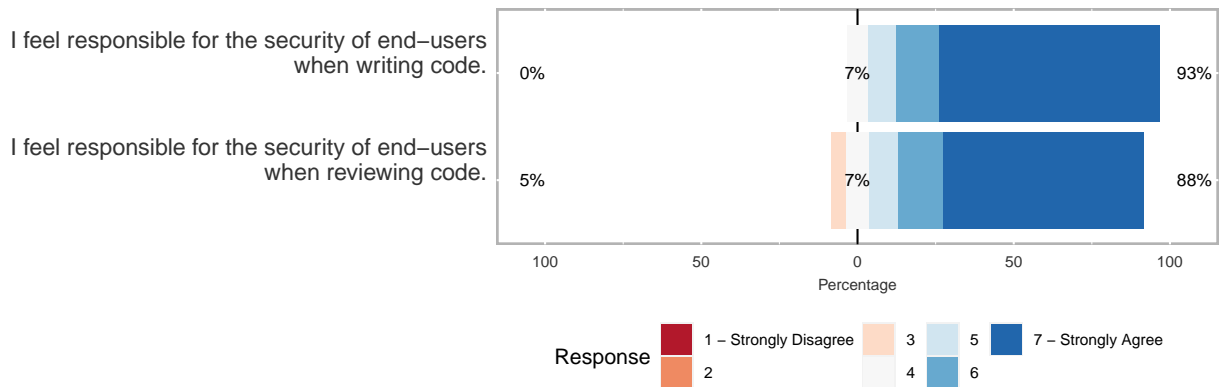


Figure 7.1.: The responses on whether the participants feel responsible for security while code reviewing and writing.

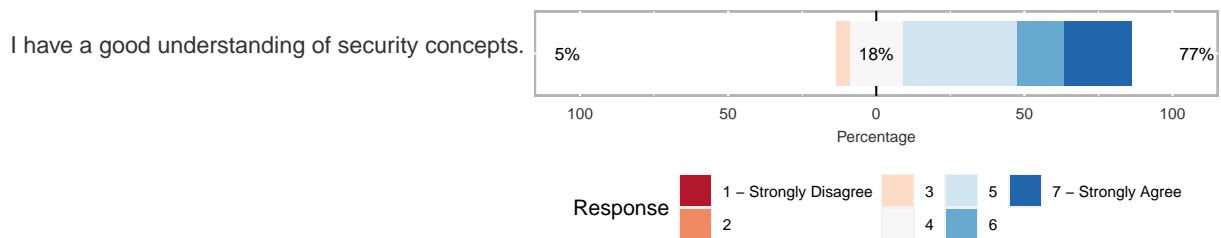


Figure 7.2.: The responses on whether the participants reported to have a good understanding of security concepts.

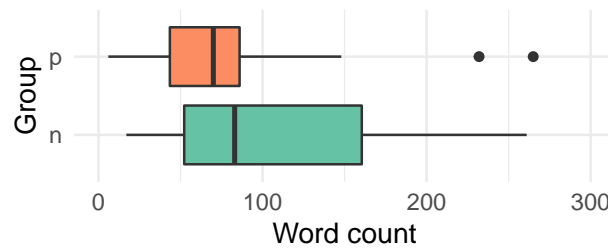


Figure 7.3.: Word count within the reviews for the prompted and non-prompted group.
p: Prompted **n**: Non-prompted

desirability bias while answering survey questions, which was also reported by Naiakshina et al. [132] in their programming study with freelancers.

Additionally, we found an overconfident self-representation of the freelance participants. The degree of agreement to the statement around the freelancers' understanding of security concepts can be found in Figure 7.2. PP1, NB2, NM2, NP1, and PM6 specifically noted that security is very important in the optional feedback field. For example, NB2 noted:

“Security is always important, developers put it in the background.”

However, 3 participants explicitly noted in the optional feedback field that security is not always important, e.g., “for robotics control” (PP8). Further, PM4 stated:

“In my experience, sometimes its not all about the security. Some occasions we have to provide hot fixes for urgent customers without thinking about the security. Yes security is an essential factor but it is not something that should be burden to a developer.”

7.3.2. Quantitative Analysis

In this section we report the results of our exploratory quantitative analysis. We tested whether prompting, programming experience or different insecure code snippets had an effect on finding the password-storage security issue. We also tested whether prompting affected the word count of the code reviews and whether the self-reported time participants spend on security differs between programming and code-review tasks.

7.3.2.1. Effect of Prompting on the Word Count

On average the reviews contained 100 words (md: 78 words) with the smallest review containing 6 words and the largest 443 words. We did not find a significant effect of prompting on the word count (Wilcoxon rank-sum, $W = 300.5$, $p = 0.17$). Figure 7.3 visualizes the word count in both groups. Both medians were in a similar range, however, the non-prompted group has a higher variable spread than the prompted group. The number of codes emerging from a code review did not necessarily rely on the word count in the review. Short reviews could cover different issues, while elaborate reviews with more details could discuss only one minor issue.

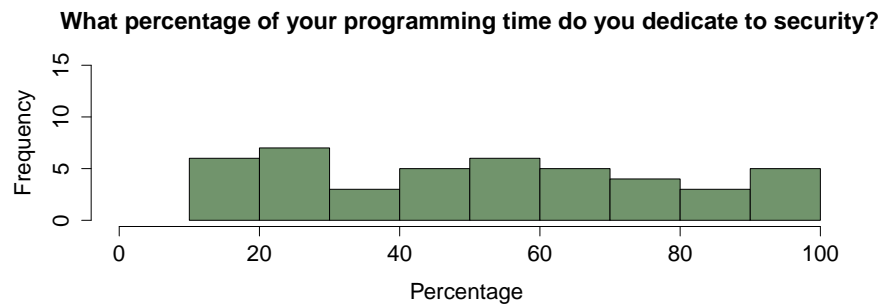


Figure 7.4.: Percentage of *programming time* freelancers dedicate to security (self-reported)

7.3.2.2. Effect of Prompting on Finding the Password Storage Issue

We evaluated whether participants correctly indicated that there was an issue with password storage security in the snippet (*found password storage issue*). We excluded participants who wrote that the passwords should be “encrypted” if no hashing function was recommended. This might mislead the developer receiving the review to implement encryption instead of hashing the passwords. Naiakshina et al. [132] reported some encryption solutions, which shows that this might be a problem. Eleven prompted and two non-prompted participants correctly stated that password storage was not solved securely in the code snippet. Thus, prompting had a significant effect on finding the issue in the code snippet (FET: $p = 0.008^*$, $cor - p = 0.02^*$, CI = [1.44, 89.85], OR = 8.28).

7.3.2.3. Effect of Experience on Finding the Password Storage Issue

In [70], Edmundson et al. did not find a significant effect of years of programming experience and whether the reviewers are more accurate or effective. We also investigated whether programming experience had an effect on whether participants found security issues with password storage. We counted how many of our 3 issues the participants were able to find (password storage, 2 distraction tasks). Similar to Edmundson et al., we did not find a significant correlation between the number of issues and the years of general experience ($r = 0.05$, $p = 0.73$). Further, we did not find a significant correlation between the number of issues and years of Java experience ($r = 0.06$, $p = 0.72$). We also did not find a correlation between Java experience and whether participants found the password storage issue ($r = 0.06$, $p = 0.71$).

7.3.2.4. Effect of Different Insecure Code Snippets on Finding the Password Storage Issue

All the three code snippets (plain text, Base64, MD5) showed an example for insecure user password storage in a database. We tested whether the different snippets had an effect on whether participants found an issue with secure password storage. For example, participants presented with an MD5 example might rather report an issue with password-storage security than participants presented with a code snippet where passwords were stored as plain text. We found, however, no significant effect within each subsample, neither the non-prompted (FET: $p = 0.07$, $cor - p = 0.14$) nor the prompted group (FET: $p = 0.26$, $cor - p = 0.26$).

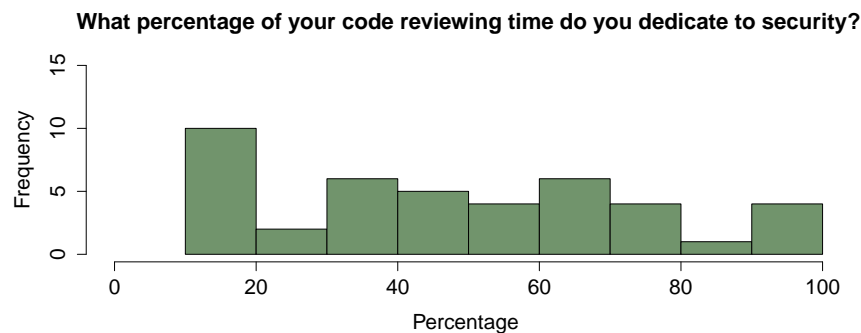


Figure 7.5.: Percentage of *code reviewing time* freelancers dedicate to security (self-reported)

7.3.2.5. Time for Security

While Figure 7.4 visualizes the distribution of percentages of *programming* time dedicated to security, Figure 7.5 summarizes the distribution of percentages dedicated to security during *code reviews*, according to our participants. The reported median percentage of time that the participants dedicated to security during code reviewing was 50 (mean: 51.64, min: 15, max: 100, sd: 26.54). The reported median percentage of time that the participants dedicated to programming was 55 (mean: 54.89 min: 10, max: 100, sd: 26.31). We did not find a significant difference between both reported time estimations using the sign-rank Wilcoxon test ($V = 247, p = 0.12$).

7.4. Discussion

RQ1: Developers' behavior in a security-critical code-reviewing task: Code reviewing is a technique applied at the end of the SDLC, used as one of the final steps by software developers to ensure programming code quality before software release. In comparison to programming code creators, developers take roles as programming code inspectors, which might increase their security awareness. Our study results showed, however, that this is not necessarily the case. It is alarming that almost half the participants wanted to release the insecure code snippets, although security issues with password storage can endanger millions of end-users' data. Even if participants indicated secure password storage as an issue, they often suggested poor techniques or weak hashing algorithms to improve the code. Such poor suggestions, however, might initiate the code creator to revise the code without really improving security. It seems freelance developers need to be reminded of security during code reviewing, otherwise they might be too focused on other issues such as logical mistakes, conventions etc.

RQ2: Factors influencing developers' security awareness: We did not find an effect between which insecure programming code snippet the participant received and whether the participant reported issues with secure password storage. This is especially interesting, considering the fact, that they not only involved plain text password storage, but also Base64 encoding and even MD5 as a hashing function. Furthermore, we did not find an effect of programming experience on finding the insecure password-storage issue, which might indicate that more experience does not

necessarily mean that the reviewers are more security aware or effective in finding security issues.

However, similar to the programming studies of Naiakshina et al. [130, 132, 133], we found that prompting for security in the reviewing task had an effect on finding the password storage issue in the code snippet. This means, that only if security requirements were mentioned in the task description, do participants consider security issues with password storage in their code reviews. Our results suggested that similar to programming, security needs to be part of the task during code reviewing as well. Therefore, we recommend to prompt for security when a code review is required.

RQ3: Developers' security responsibility in a code review: Our participants reported to spend half their programming and code reviewing time on security. We did not find a significant difference of the reported time spent on security between programming and code reviewing. Additionally, almost all the participants indicated to strongly agree with the statement that they feel responsible for security during programming and code reviewing and that they have a good understanding of security concepts. 66% of the participants also indicated to have ensured that the user passwords were stored securely after their code reviewing task. However, considering that only 13 of 44 (30%) participants reported a security issue with password-storage and the fact that almost all of them were prompted, this might indicate a social desirability bias in surveys. Our qualitative analysis showed that a number of participants had misconceptions and outdated knowledge of secure password storage. This might also suggest that APIs and libraries need to provide safe security defaults instead of requiring software developers to choose security mechanisms.

RQ4: Methodological implications: There is only limited knowledge of using code reviewing as a methodology for security studies with developers. While we provide insights into freelancers' behavior in code-reviewing tasks, we also wanted to explore which advantages, disadvantages and parallel insights a code-reviewing study can have in comparison to a programming study with developers. While we cannot conduct a direct comparison to the study of Naiakshina et al. [132] due to methodological differences, we still discuss the methodology of code reviewing for developer security studies by comparing the advantages, the disadvantages and some parallel insights of both the study types.

One disadvantage was the lack of certain information. We were not able to calculate the security scores of participants in such detail as Naiakshina et al. did. We could not find all information for the security scores in the reviews since participants simply did not mention them. Checking whether participants found the password storage issue was, however, still possible.

Moreover, we found that prompting had an effect on participants' solutions. This indicates that researchers investigating the security awareness of freelance developers might not need to hire them for longer programming tasks. Short and focused code reviews can offer similar results. With a median of 83 minutes to complete the survey, our participants required less time than Naiakshina et al.'s participants, who worked about 6-8 hours on the programming tasks.

Furthermore, code reviewing tasks can give indications to problems with code writing. Similar to Naiakshina et al., we were able to identify different issues developers experienced with password storage. For example, MD5 and encryption were often mentioned as adequate solutions to solve the password-storage issue. However, MD5 is an outdated hash function, which is not recommended

any more for secure password storage. With encryption, participants might have referred to symmetric encryption [132], which is, as mentioned before, a discouraged practice for secure password storage. This suggests that code reviewing studies can offer valuable insights into participants' security behavior. We acknowledge though, that code reviewing is a different process to writing code and therefore it is not possible to prove which suggested solutions to the issues developers would really implement.

Similar to Naiakshina et al.'s password-storage study with students, we found that "security knowledge does not guarantee secure software" [130]. Although only 30% of our participants indicated that the user passwords were stored insecurely, 63% were able to provide a correct definition for hashing and salting after their code reviewing task. We have to note, however, that we had only limited possibilities to prove that their definitions were not simply copied and pasted from the Web. It seemed that 8 participants copied the entire definition or parts of their definition from the Internet, which indicated that knowledge questions should be treated with caution in surveys.

To sum up, we found that security prompting had a significant effect regardless of whether participants completed a programming or a code-reviewing task on password storage. Additionally, we were able to identify participants' misconceptions and outdated knowledge about secure password-storage and which criteria they believe are important in programming code. One disadvantage, however, was that we were limited in the comparison of the participants' security scores, which Naiakshina et al. introduced in their programming study. In our study, the code reviews did not offer enough details to calculate them. Still, code reviewing tasks can help investigate programming knowledge and decrease the time developers need to spend on a task. Participants needed less time to complete the study compared to the studies of Naiakshina et al. while still finding similar results with regard to security awareness and security prompting.

While we do not argue to replace programming tasks with code-reviewing tasks in security developer studies, funding is often limited within academia and smaller tasks yielding similar effects could enable more future research with developers. Therefore, we encourage the community to conduct further research into this line of work.

7.5. Summary

We conducted an online code reviewing study with 44 freelance developers showing each of them an insecure password storage code snippet. We investigated how participants behave in a code-reviewing study by considering which criteria they base their reviews on, whether they would find the security issue and most importantly, whether they would release the insecure code snippets. Additionally, we explored different factors, which might influence their behavior. For example, we explored the effect of prompting for security in the task on whether participants reported password storage security issues within their code reviews. We also explored whether participants feel responsible for and how much time they dedicate to security. Finally, we discussed the methodological implications of a code reviewing study for developer security studies.

Not even one third of our participants reported the security issue with password storage. Almost all the participants who reported an issue were prompted for security. Thus, prompting had a significant effect on participants' behavior. Still, almost half the participants wanted to release the code as it is, which is alarming since insecure password-storage is a major issue endangering

millions of users. Finally, our findings suggest that code reviewing studies could be an interesting approach for conducting security developer studies.

For future work we recommend testing a hybrid between a code reviewing and a code writing developer study: a participant could receive functional insecure code and be asked to write a review and if necessary to correct the issues within the code. This could combine the advantages of both the methodologies. However, it might also increase the time of solving the study for the participants again.

8. Conclusions

Security vulnerabilities of software threaten the data security of its end users. Software developers can develop secure code to prevent these vulnerabilities from occurring, but developers can have their misconceptions and they make mistakes, just like end users. To investigate these misconceptions and errors, and to help prevent and correct them, more security studies with software developers need to be conducted. This thesis presented five studies with software developers. Besides investigating secure programming, the studies also offered several methodical insights into strategies for conducting developer studies. Different stages in the study design and evaluation were considered, and the research questions aimed to provide more insights for future research. To spare expenses for researchers, methodological recommendations on how to conduct security studies with software developers were presented in this thesis. While the comparison studies still only hold true for those specific use cases, they offer first indications and impetuses for HCI researchers.

First, options to filter out participants without programming skill were investigated, the goal being to prevent detrimental noise in survey data in security studies with software developers. We developed screener questions that are easy and quick to answer for programmers, but which non-programmers find challenging. To evaluate our questionnaire for efficacy and efficiency, several batches of participants with and without programming skill answered the questions. Finally, an adversarial setting was created to test the questions that performed best. We recommended questions that researchers can use to screen participants in security studies with software developers. Further, we conducted a follow-up study which investigated screener questions with time limits to minimize the time needed on the screener while improving data quality.

Furthermore, we conducted a study to test for deception in announcing a study as such, instead of disguising it as a real job. As deception needs to be used with careful consideration, we replicated our study [132] but announced it as a study. Our findings suggest that for that particular security study, deception did not show indications for different effects, and the open recruitment without deception appeared to be a viable recruitment method.

Further, we tested code-reviewing as a new methodology for security developer studies in online surveys. We found that security awareness remains an issue for developers during code-review. While we provide insights with regard to the security awareness of freelance developers in a code reviewing task, we also compared our results with similar password storage studies from related work. Finally, we discussed code reviewing as a new methodology for future research.

Concerning the evaluation of security studies with software developers, we conducted a qualitative and quantitative study on security warnings with developers. We found that the key results from the qualitative analysis were only confirmed by the quantitative analysis. This study, along with another study of ours [131], provides indications that qualitative analysis is a valuable method in security studies with software developers, and it can uncover key results.

Overall, our methodological insights are the following:

- **Data validity: Screening.** As presented in Chapter 4, we found that screening in online surveys is required to improve the data validity of the study data. Since recruitment services [7, 1] offer services to help with recruiting developers as survey participants, it is important to filter out participants without programming skill. Further, the follow-up study (Chapter 5) showed

that screeners with time limits were even more efficient. The pool of screeners was extended. We provided recommendations on how to improve the process. With this work, researchers can easily screen in their studies and filter out participants with no programming skill. Our pool of questions can be extended in the future to prevent cheating even further.

- **Ecological validity: Announcing a study.** In Chapter 6, we investigated whether announcing a study as such yields different effects when compared to a study disguised as a real job. Our results indicate that transparently announcing a study remains a viable option for researchers conducting developer studies. Since deception can bear issues, this study serves as one data point stating that deception in that particular step might not be necessary in security studies with software developers.
- **Tasks: Code-reviewing vs. Code writing.** Our study showed that different tasks can lead to similar results in developer studies. Code-reviewing as a study methodology can help in gaining insights into the knowledge and process of reviewing a written example. Chapter 7 summarizes the results of our study. The quantitative results confirmed the key findings of the qualitative analysis.
- **Qualitative vs. Quantitative.** We found indications that a qualitative analysis can lead to similar key results as a follow-up quantitative analysis. Our study and previous work had already suggested this. Since recruiting developer in high numbers remains a challenging task, we suggest that in some scenarios a qualitative analysis offers similar key points as a quantitative investigation. More research through more developer-related studies is needed to obtain more data to prove this. Chapter 3 provides more detailed insights into that. While the primary goal of this study was to investigate the perception of software developers on security warnings, we can also draw methodological conclusions from this study. The qualitative analysis hinted at points that were later proved in the quantitative analysis. For example, there was no clear winner emerged among the warning types and no clear preference for the time of display was evident. Interestingly, the quotes from the interviews often provided us with insights into the reasoning behind the choices and answers of the participants. This study is thus one indicator that more qualitative work can be done in conducting software developer studies. In some cases the qualitative approach can yield finer details into the thoughts of developers.

More research into the methods used in security studies with software developers is required to aid researchers in the design of studies of this nature. Just as the end user has to be considered in usable security and privacy research [15], security studies with software developers have to be carefully investigated to prevent end users from incurring massive data losses in the future.

To provide more recommendations, more comparison studies have to be conducted by the research community. Our studies provide initial indications of several nuances appearing during study design. More research needs to be conducted in terms of task design. Different task lengths could yield different results with regard to security awareness. Further, different code-reviewing task designs could be investigated to find out more about code-reviewing as a task in security

studies with software developers. Examples could refer varying task lengths, as well as more specific task descriptions.

A. Appendix: Analysis: Security Warnings for Software Developers

A.1. Structure

Our appendix is structured as follows. In Section A.2, we provide information on our qualitative study and in Section A.3, we provide information on our quantitative study.

The **qualitative study** material covers

1. the **semi-structured guideline** including a description and a reference to each type of warnings, (Section A.2.1)
2. **participants' demographics** (Section A.2.2),
3. the **codebook** (Section A.2.4.1) and,
4. **themes** extracted through our Grounded Theory (GT) approach (Section A.2.4.2 and Figure A.1).

The **quantitative study** material covers

1. the **visual representation of the variations of warning types** in the online survey (Section A.3.1),
2. the **invitation text** to our survey (Section A.3.2),
3. **survey questions** including the **programming test** (Section A.3.3),
4. **participants' demographics** (Section A.2.2 and A.2.3) and,
5. **participants' rating of the different types of warnings** (Section A.3.5).

A.2. Qualitative Study

Participants were presented different types of warnings. A generic IDE drawing was compiled for all warnings, because participants were assumed to have experiences with different IDEs. Additionally, the study's focus was on warnings in general, instead of the details differing between IDEs.

A.2.1. Semi-structured Guideline

Please state who you are and what kind of development work you do. Please also tell us what editor or IDE you use for your work. We want to investigate which warnings types would be helpful. We will present different warning approaches and we would like to hear your opinion on each of them.

- **Compiler Warning.** The first is the compiler warning. In foobar, something happens that triggers a warning. The warning is displayed in the command line output in the IDE (see Figure A.15).
- **Warning Marker (Yellow).** After that, we have two markers for the warning. The first is the usual yellow marker. As you can see, the function is underlined in yellow, and there is a small caution button besides the line number (see Figure A.16).
- **Warning Marker (Own Color, Blue).** Secondly, we could define an own color, we chose blue and a small lock symbol to indicate that it is a security warning (see Figure A.17).

- **Pop-up Warning.** Here, you see the pop-up warning. The warning would be displayed in a pop-up as soon as it is triggered (see Figure A.18).
 - **Plugin Warning.** Currently, there are already plugins which display warnings. To do so, they often use a violations outline in a corner of the IDE. Further, an arrows marks the warning priority (see Figure A.19).
 - **Security View.** Another warning type is a security warning view in the IDE and an own button just like the debug or run button (see Figure A.20).
 - **Warning on Committing.** Another use case would be to warn a developer when committing code. In this case, a pop-up appears (see Figure A.21).
 - **[Added:] Warning on Committing: Automated Security Ticket.** Finally, this is a committing warning where you would generate an automatic security ticket (see Figure A.22).
-
- Which warning types would you use? Why?
 - (For each warning type:) What do you think about this approach?
 - When should this warning be displayed? (During implementation/after implementation/before committing.)
 - How often should this warning be displayed? (Once/more often:...)
 - Could you think of a better approach?
 - *Security responsibility: [Added for professional developers]*
 - Is there a person responsible for code security in your company? Do you have a security responsible person in your company who you could consult?
 - Do you feel responsible for the security in code? While coding, when do you think about security? Is security is part of the process?
 - Would you like to delegate a warning to another developer?
 - *Experiences with breaches: [Added for professional developers]*
 - Have you already experienced security breaches in the past? Which ones?
 - If yes: Do you think one of these warnings would have prevented the security breach?
 - Are you familiar about warnings, or have you had any experience with them while programming?
 - Do you have examples of cases in which security warnings could be useful?
 - There are different cases where security warnings could be triggered to inform the developer that they are programming potentially non-secure code. Could you think some examples?
 - Do you think security warnings for developers could be useful?
 - Which color would you prefer for security warnings?
 - Would you like warnings to be deactivated in case of a test project ?

A.2.2. Demographic Questions

- Please select your gender. *Female/Male/warnings/tables/Prefer not to say*
 - Age: *[free text] years*
 - What is your current occupation? *Freelance developer Industrial developer/ Industrial researcher/ Academic researcher/ Undergraduate student/ Graduate student/ Other: [free text]*
 - Do you have a university degree? *Yes/No*
-

- *Primarily for students:* Currently, do you have a part-time job in the field of Computer Science? If yes, please specify: [free text]
- At which university/universities were/are you enrolled? [free text]
- Were/Are you taught about IT-security at your university? Yes/ No / I don't recall
- Which security lectures did you pass in your Bachelor's/Master's program? [free text]
- Were/Are you taught about IT-security extramural? Yes/ No
- What was your main source of learning about IT-security? [free text]
- How did you gain your IT skills? [free text]
- How did you gain your IT-security skills? [free text]
- What type(s) of software do you develop? Web applications/ Mobile applications/ Desktop applications/ Embedded applications/ Enterprise applications/ Other (please specify)
- Which programming languages do you know? [free text]
- How many years of experience do you have with software development in general? [float] years
- Which IDEs do you use? [free text]
- Which tools do you use for software development? [free text]
- What is your nationality? [free text]
- Thank you for answering the questions! If you have any comments or suggestions, please leave them here: [free text]

A.2.3. Demographics of Participants

Our participants reported to have used or currently use the following IDEs: Eclipse [69], Netbeans [134], IDEA IntelliJ [104], VisualStudio [125], PyCharm [106], Sublime [120], CLion [103], PhpStorm [105], Android Studio [167], Spyder [55], Notepad++ [6], Code::Blocks [9], Geany [90], ABAP Workbench [139], Google Colaboratory [10].

Also the fields in which our participants had worked in were quite diverse: system oriented, networks, SAP-development, machine learning, monitoring, databases, web development, open source projects, security, mobile development etc.

Further, our participants reported to be familiar with different programming languages: Java (24), Python (22*), C/C++ (18), JavaScript (6*), PHP (6*), C# (3*), ABAP (3*), Go (2*), Pascal (2*), Haskell (2*), Rust (1*), VisualBasic (1*), Cobal (1*), Groovy (1*), Perl (1*), Lisp (1*) and Ruby (1*). The symbol * indicates "out of 33 participants."

A.2.4. Grounded Theory Analysis

A.2.4.1. Codebook

In Table A.1 we present our codebook with at least one example quote from the interviews per category.

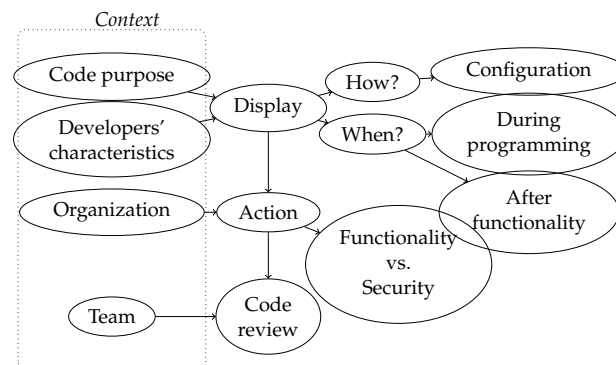


Figure A.1.: Themes extracted using grounded theory.

A.2.4.2. Themes

As suggested by Charmaz [46], we used the idea of theoretical sorting to develop a diagram out of our memos, categories and themes. Figure A.1 provides an overview of the themes that came out of the grounded theory study. Categories and memos have informed the themes. The following themes have evolved out of the interviews. Occurring themes could change according to the theoretical sample approach within the sample process:

1. Student participants and freelance developer:

- **Functionality first, security second.**
 - Committing and pop-up security warnings are preferred after finishing the programming functionality.
- **Security warnings during programming and after finished programming.**
 - Committing and pop-up security warnings are preferred after finishing the programming functionality.
 - Warning marker are desired during programming.
 - Security view should be available at any time.
- **Disabling security warnings depends on use-cases.**
 - Some believed disabling should not be allowed for security warnings and others believed it depends on situations. For instance, if developers only test their program, they do not want to be displayed security warnings.
 - However, disabled security mode should always be marked!
- **Extra view lead to an additional overhead.**
 - Participants believed that too many views could be disturbing for them, because already a lot of views exist in the IDEs.
- **Security deserves an extra warning, e.g., with an own color.**

2. Security start-up participants:

- **Functionality and security equivalent important**

- No merge requests should be allowed if security issues are not fixed. Additionally, others should see the security issues on merge requests as well.
 - Wish to involve team members to fix security issues.
 - Security ticket on committing requested.
 - Warnings if pulling security issues.
 - **Security warnings during programming and after finished programming.**
 - Combination of warning on committing and warning marker with an own security color and symbol preferred.
 - **Disabling security warnings depends on use-cases.**
 - Security warnings are important, but it should be possible to disable them in some use-cases or in a testing scenario.
 - **Extra view could lead to an additional overhead.**
 - Participants believed that too many views could be disturbing for them, because new views always require familiarization each time.
 - **Quality (security) costs time.**
 - **Developer the weakest link.**
 - It depends on the developer how secure APIs are.
 - **Workflow integration important.**
 - Developers using the command line would not see security warnings in an IDE.
 - Pop-ups intrusive.
 - **Warnings keep developers up-to-date.**
3. Academic focus group participants:
- **Trade-off between functionality and security.**
 - On the one hand, developers should be warned about security issues in the moment when they arise. On the other hand, developers should not be kept from working.
 - Security interrupts the development process.
 - **Testing mode could be insecure.**
 - Testing mode without security warnings should only be available for security aware developers.
 - Developer should be warned that her/his security warnings are disabled when leaving the testing mode.
 - **Configuration.**
 - There will be never one perfect warning system. It depends on habits, experience and different views.
 - Pre-defined preference profiles of warning types should be suggested to developers.
 - Pop-ups only if requested and for “are you sure” questions.
 - **Security responsible person in teams requested.**
 - **Text completion (secure parameter defaults) for security requested.**
4. Government institution participants:
- **Functionality first, security second.**
-

- After finished programming. Reasons: Security requires time, time pressure of the company, security not demanded by the company.
- **Warnings during programming and after finished programming.**
 - During programming. Reasons: could be too late otherwise, developers rather fix issues rather if they are shown immediately (during programming).
 - After finished programming. Reasons: security requires time, time pressure of the company, security not demanded by the company.
- **Disabling security warnings depends on use-cases.**
- **Success of the security ticket on committing warning depends on the team size.**
 - Unclear responsibility.
- **Security team/policies in company do not have knowledge in programming and nobody checks code for security.**
- **Specific guideline for specific project requested.**
 - Security policy guidelines of the company were not developed from tech people.
 - Developers overloaded with information.
- **Configuration.**
 - Like the idea of having a list with warnings: issues, which are solved can be deleted and others will stay there.
 - Pop-up intrusive if working with key board.
 - Security view good as a summary for security issues found by different tools.
 - Would like to get additional information (source) for security warning.
 - Warning marker should be shown within source code.
 - Committing: idea of being reminded at the end of the working process about security issues developers might forgot or have overseen; “stop sign”.
 - Combination of security marker warning and the security view warning.

5. Financial sector participants:

- **Company decides about the trade-off between functionality and security.**
 - Participants perceive functionality and security as equally important.
 - Security checks could prevent from working.
 - Company with no demand for security, does not force to consider security while programming.
 - **Security warnings after finished programming.**
 - For workflow integration, security is prioritized at the end.
 - **Configuration.**
 - Companies with security context prefer the security ticket warning and would like to disable warnings in appropriate use-cases.
 - **Good security warnings important.**
 - For end-user software outside world.
 - Current warning systems prevent from developing.
 - **Better education instead of security warnings.**
-

- **Team integration important.**
- **Code review important matters.**
- **Security team/policies in company do not have knowledge in programming and nobody checks code for security.**

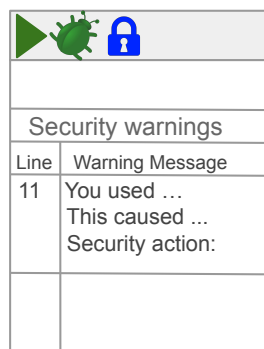
A.3. Quantitative Study

After our qualitative analysis, we built a quantitative survey to be able to quantify our results. Usually all multiple choice responses were shown randomized to our participants.

A.3.1. Types of Warnings

The different types of warning variations were presented to our participants:

- **Security View:** see Figure A.2.
- **Warning on Committing with Security Ticket:** see Figure A.3.



Security warnings	
Line	Warning Message
11	You used ... This caused ... Security action:

Figure A.2.: Security view

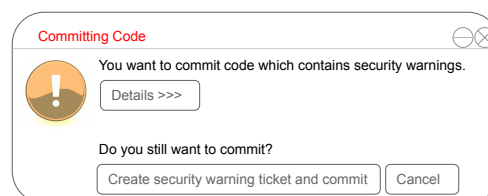


Figure A.3.: On committing with security ticket

A.3.2. Invitation

Receive a 20 euro Amazon voucher by answering a 10-15 minutes survey! We are researchers from the University of Bonn and are looking for motivated software developers who will take part in a 10-15 minutes survey. The survey is about a topic in software development and is conducted

in English in order to ensure an international comparison. We look for about 25-30 participants. Interested? Find out whether you are a suitable candidate for our survey by answering two questions: [LINK](#)

A.3.3. Survey

- In order to make sure you are a suitable candidate for our research, we would like you to answer three questions before starting with the study.

How old are you? [free text]

- Where are you employed?

- Africa
- Asia
- Australia
- Canada
- New Zealand
- UK
- USA

```

1   main{
2       print(func("hello world"))
3   }

5   String func(String in){
6       int x = len(in)
7       String out = ""
8       for(int i = x-1; i >= 0; i--){
9           out.append(in[i])
10      }
11      return out
12  }
13

```

Figure A.4.: Test for Software developing skills

- Please select the returned value of the pseudo code above[see Figure A.4]:

- hello world hello world hello world
- world hello
- hello world 10
- HELLO WORLD
- hello world
- dlrow olleh

- **Introduction:**

We are researchers from the University of Bonn and are investigating security warnings for

developers. There are tools such as static analyzers which can inform developers about security issues in their code. To do so, these tools need to display warnings to the developer. We are investigating how and when security warnings can be displayed in an Integrated Development Environment (IDE). By taking part in our study you will help us understand when and how developers would prefer to be warned about security issues in their code. For this we will show you different types of warning and ask for your opinion on these. There are no right or wrong answers since we are purely interested in your opinion.

- **Consent:** I have read and understood the informed consent form. I consent that the data from the survey can be used for research purposes. Researchers will have access to this data only if they agree to preserve the confidentiality of the data and if they agree to the terms specified in the consent form. Only anonymised data will be published.
 - I consent
 - Which IDE (integrated development environment) do you use most?
 - Netbeans
 - Eclipse
 - Microsoft Visual Studio
 - IntelliJ IDEA
 - Other:
 - **Compiler Warning:**

Below you see an example of how a security warning could be presented by the compiler. The warning would be displayed in the command line output in your IDE .
 - Please state your agreement to the following items: 1: *Strongly Disagree* - 7: *Strongly Agree*
 - I would like to be informed about security issues in my code with this type of warning.
 - I would be quickly annoyed by this type of warning
 - It would be easy to overlook this type of warning.
 - IDEs already have too many warnings of this type.
 - I am familiar with this type of warning.
 - **Markers:**

Below you see examples of how markers can be used to present a warning. The warning is displayed within the code editor of the IDE. The piece of code triggering the warning is underlined and there is an icon next to the line of code. Different colours can be used for the underlines. Yellow markers are commonly used to signify warnings and red is used to signify errors. Security warnings could also use these colours or use their own colour. You can hover over the icon to get more information about the warning.
 - In which colour would you like your IDE to underline code to highlight security warnings?
 - Black
 - Brown
 - Blue
 - Red
 - Yellow
 - Green
-

- Grey
 - Pink
 - White
 - Violet
 - Orange
 - Other:
- Assuming the IDE uses the colour you chose above: Please state your agreement to the following items.*1: Strongly Disagree - 7: Strongly Agree*
 - I would like to be informed about security issues in my code with this type of warning.
 - I would be quickly annoyed by this type of warning
 - It would be easy to overlook this type of warning.
 - IDEs already have too many warnings of this type.
 - I am familiar with this type of warning.
 - Pop-up:
Below you see an example of a pop-up warning. The warning would be displayed by your IDE as soon as you complete a code statement that triggers a security warning.
 - Plugin:
Below you see an example of how an IDE view can be used to show warnings. Such a view commonly already exists to give an overview of non-security/general warnings and errors. Security warnings could be added to this view or receive a dedicated security warnings view. You can click on the security-warning to receive additional information.
 - I would prefer...(only one selection possible)
 - security warnings to be displayed in a dedicated security warnings view.
 - security warnings to be displayed in the same view as the general warnings and errors.
 - For the view you selected above: Please state your agreement to the following items.*1: Strongly Disagree - 7: Strongly Agree*
 - I would like to be informed about security issues in my code with this type of warning.
 - I would be quickly annoyed by this type of warning
 - It would be easy to overlook this type of warning.
 - IDEs already have too many warnings of this type.
 - I am familiar with this type of warning.
 - Warning on Committing:
Below you see an example of a pop-up security warning which would be displayed in your IDE when you try to commit code which contains a security issue.
 - Please state your agreement to the following items.*1: Strongly Disagree - 7: Strongly Agree*
 - I would like to be informed about security issues in my code with this type of warning.
 - I would be quickly annoyed by this type of warning
 - It would be easy to overlook this type of warning.
 - IDEs already have too many warnings of this type.
 - I am familiar with this type of warning.
-

- Warning on Committing Ticket:
Below you see an example of a pop-up security warning which would be displayed in your IDE when you try to commit code which contains a security issue. If the developer proceeds with the commit, a ticket is created to keep track of the issues.
 - For whom should the ticket be created? *Someone responsible for security issues; Myself; I don't know what a ticket is.; Other:* Assuming the ticket is created for the person you selected above: Please state your agreement to the following items. *1: Strongly Disagree - 7: Strongly Agree*
 - I would like to be informed about security issues in my code with this type of warning.
 - I would be quickly annoyed by this type of warning
 - It would be easy to overlook this type of warning.
 - IDEs already have too many warnings of this type.
 - I am familiar with this type of warning.
 - When would you like to be warned about security issues in your code? (Multiple answers possible)
 - While coding - right away
 - While coding - after completing a function
 - While coding - in regular intervals
 - Before running
 - Before committing
 - Before release
 - On demand (e.g. by clicking a button or opening a view)
 - Never
 - Do you think an IDE should offer different kinds of warnings so developers can choose their preferred way of being warned? *Yes; No*
 - If you are using or in the past have used a static analyser please select the one you have the most experience with. If you have never used a static analyser please select None.
 - Checkmarx Static Code Analysis
 - Clang
 - CodeSonar
 - FindBugs
 - Fortify
 - PMD
 - None
 - Other:
 - Please state your agreement with the following statements with respect to the static analyser you have the most experience with. *1: Strongly Disagree - 7: Strongly Agree*
 - It was easy to collaborate as a team when using the tool.
 - The output results of the tool usually offered me a fix for the problem.
 - The output results of the tool were easy to understand.
 - I feel that the tool helped me with my programming work.
-

-
- The amount of output of the tool was reasonable. The percentage of false-positive was acceptable.
 - I would recommend this tool to other programmers.
 - It was possible to customize the tool.
 - Using the tool could be easily integrated into my workflow.
 - Do you currently use static analyzers or other code checking tools? *Yes; No*
 - What are the reasons you don't use / stopped using static analyzers or code checking tools? (Multiple selection possible)
 - I don't know how to use them
 - They are too annoying
 - I don't know any tools
 - Using them is too time consuming
 - I am not interested in security
 - I never thought about using tools
 - They are not effective
 - I am not responsible for code security
 - They are too expensive
 - Other:
 - Please state your agreement: *1: Strongly Disagree - 7: Strongly Agree*
 - I would like to be able to snooze security warnings.
 - Security issues should be treated as blocking errors and not warnings.
 - Security issues should block merge requests.
 - I would like to be able to turn off security warnings entirely.
 - This is an attention check question. Please select "1" as your answer.
 - I would like to be able to disable all security for periods of time (e.g. prototyping stage)
 - In your current position how many security warnings do you see while programming?
 - More than 5 a day
 - Less than 5 a day
 - Less than 5 a week
 - Less than 5 a month
 - Less than 5 a year
 - none
 - Who do you think is responsible for fixing functionality bugs?
 - Me
 - Someone else:
 - Who do you think is responsible for fixing security bugs?
 - Me
 - Someone else:
 - Please select what is more important ...(Slider between Functionality and Security , Equally important at 50%, Not Applicable)
-

- ... to you:
 - ... to your organization:
 - ... your team:
 - ... your direct supervisor:
 - Does your organization have security coding policies? *Yes; No*
 - Is there a person or team in your organization who is responsible for code security? *Yes; No*
 - Are there colleagues who you can ask on an informal base in cases of security issues in your code? *Yes; No*
 - How are your project team's total software security efforts divided among the following stages? [Must add up to 100]
 - The design stage
 - While implementing the code
 - During testing by developers
 - During code analysis (e.g. using static analysis tools)
 - During code review
 - During testing that is done by someone other than the code owner
 - Have you already experienced a security incident *Yes;No*
 - involving code you created?
 - involving code created by a colleague?
 - involving code created by a third party which was integrated into your software?
 - I have a good understanding of security concepts. *1: Strongly Disagree - 7: Strongly Agree*
 - Please rate the following items: *1: Never - 7: Every time*
 - How often do you ask for help when faced with security problems?
 - How often are you asked for help when others are faced with security problems?
 - How often do you need to add security to the software you develop?
 - What percentage of your programming time do you spend on security? (0-100)
 - **Demographics:**
 - Age: .. *years*
 - Please select your gender. *Female/Male/Prefer not to say/ Diverse:*
 - What is the highest education level you have completed?
 - High school graduate (high school diploma or equivalent including GED)
 - Some college but no degree
 - Associate degree in college
 - Bachelor's degree in college
 - Master's degree
 - Doctoral degree
 - Professional degree (JD, MD)
 - None
 - In which country do you mainly work? *[free text]*
 - What type(s) of software do you develop? *Web applications/ Mobile applications/ Desktop applications/ Embedded applications/ Enterprise applications/ Other (please specify)*
-

- How many years...
 - * .. of programming experience do you have?
 - * ... have you been working in a job where software development was a substantial part of your activity?
 - * ... was security relevant for your programming work?
- How many employees work in your organization?
 - 1-4
 - 5-9
 - 10-19
 - 20-49
 - 50-99
 - 100-249
 - 250-499
 - 500-999
 - 1000 or more
 - I don't work in any organization.
 - I prefer not to answer.
- How many people work in your team? Please enter 1 if you work on your own.
- What is your current occupation?
 - Freelance developer
 - Industrial developer
 - Industrial researcher
 - Academic researcher
 - Undergraduate student
 - Graduate student
 - Other: [free text]
- Thank you for taking part in our study! We really appreciate your time and effort. We hope our results will help improve how IDEs assist developers in their work. If you have any comments or suggestions, please leave them here and then please click on "Continue" to complete the survey.

A.3.4. Demographics of Participants

IDEs

32% (16/50) reported to use Eclipse, 24%(12/50) used IntelliJ IDE and 24%(12/50) Microsoft Visual Studio. 6 % indicated to not use any IDE right now, 6% used Pycharm, 1 participant used Jade, 1 participant used Visual Studio Code and 1 participant reported to use an internal IDE. Finally, one participant used monodevelop.

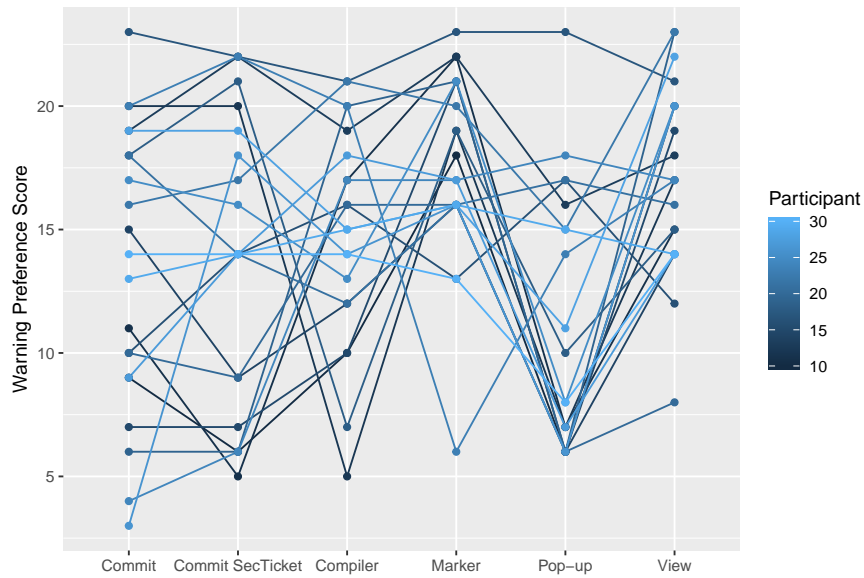


Figure A.5.: Preference score for 20 random participants [batch 2]

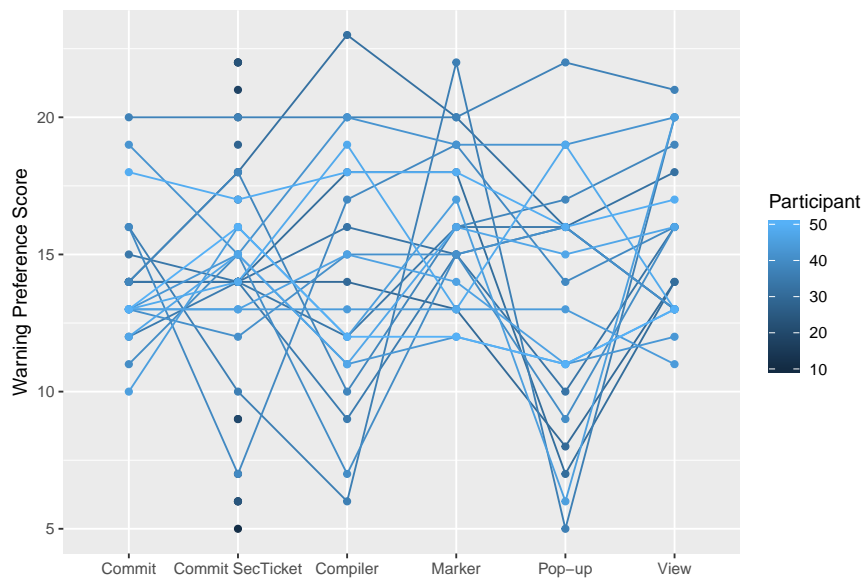


Figure A.6.: Preference score for 20 random participants [batch 3]

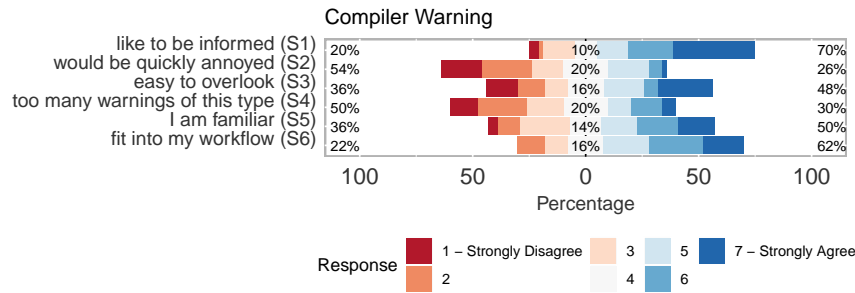


Figure A.7.: Compiler warning

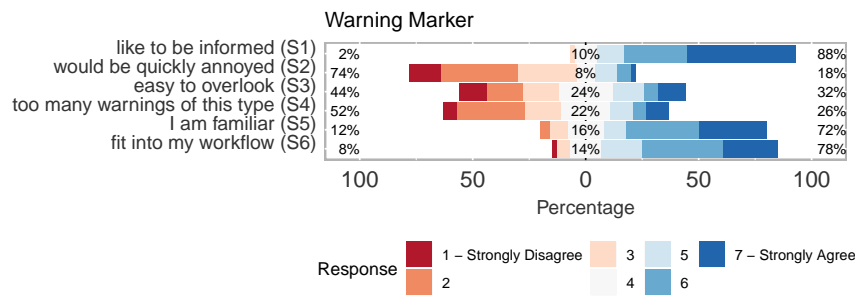


Figure A.8.: Warning marker

A.3.5. Warning Rating

Participants rated warnings on a 7-point Likert scale. The resulting rating can be found in Figure A.7 for **compiler warnings**, in Figure A.8 for **warning marker** warnings, in Figure A.9 for **warning view** warnings including general and security warnings, in Figure A.10 for **warning view** warnings including **only security warnings**, in Figure A.11 for **pop-up warnings**, in Figure A.12 for **warnings on committing** and in Figure A.13 for **warnings on committing with security ticket**. Figure A.14 shows the statements regarding ignoring security warnings.

In the figures A.5 and A.6 we plotted the preference score ranking of 20 random participants for each type of warning. Each spike in the graph indicates a preference jump within one participant.

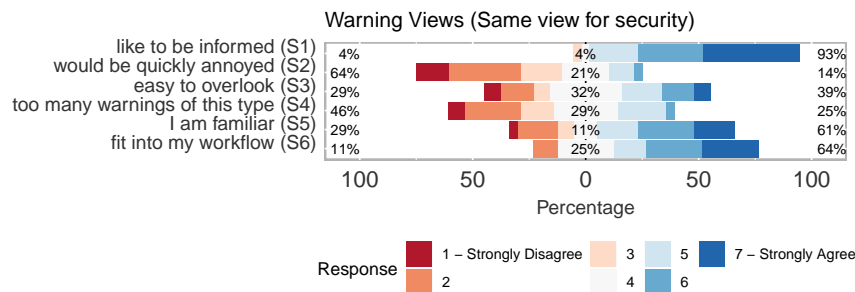


Figure A.9.: Warning view including security warnings

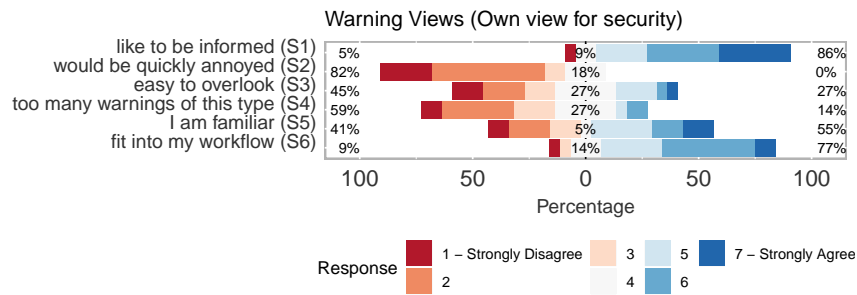


Figure A.10.: Warning view only for security warnings

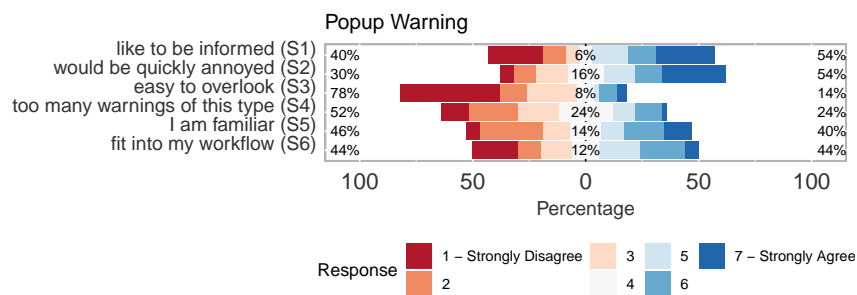


Figure A.11.: Pop-up warning

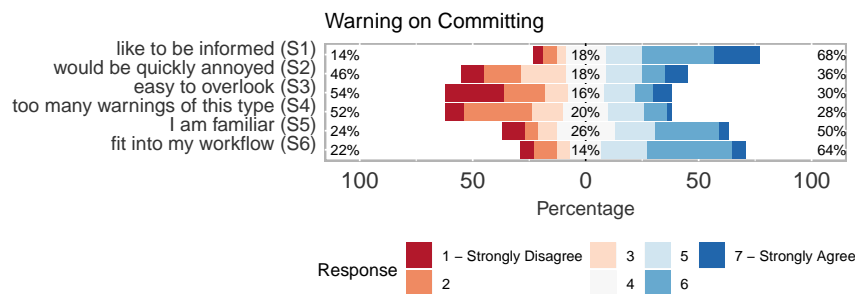


Figure A.12.: Warning on committing

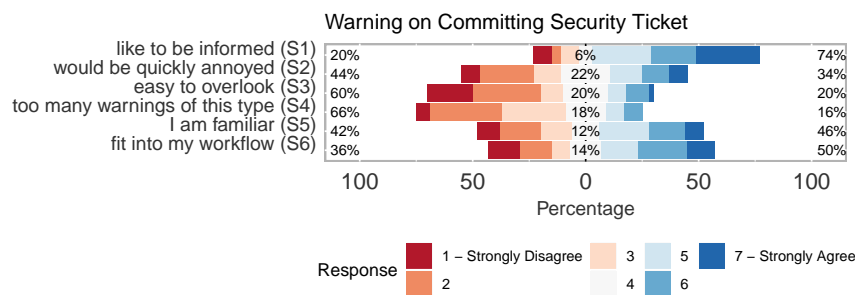


Figure A.13.: Warning on committing security ticket

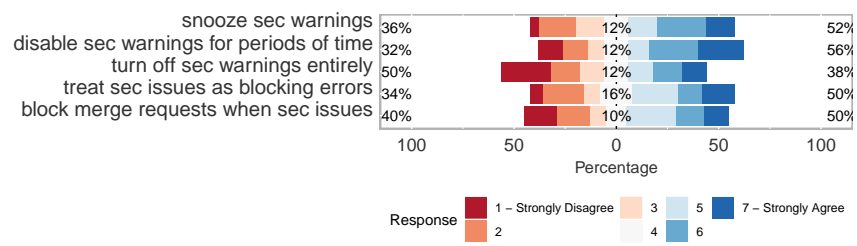


Figure A.14.: Ignoring security warnings

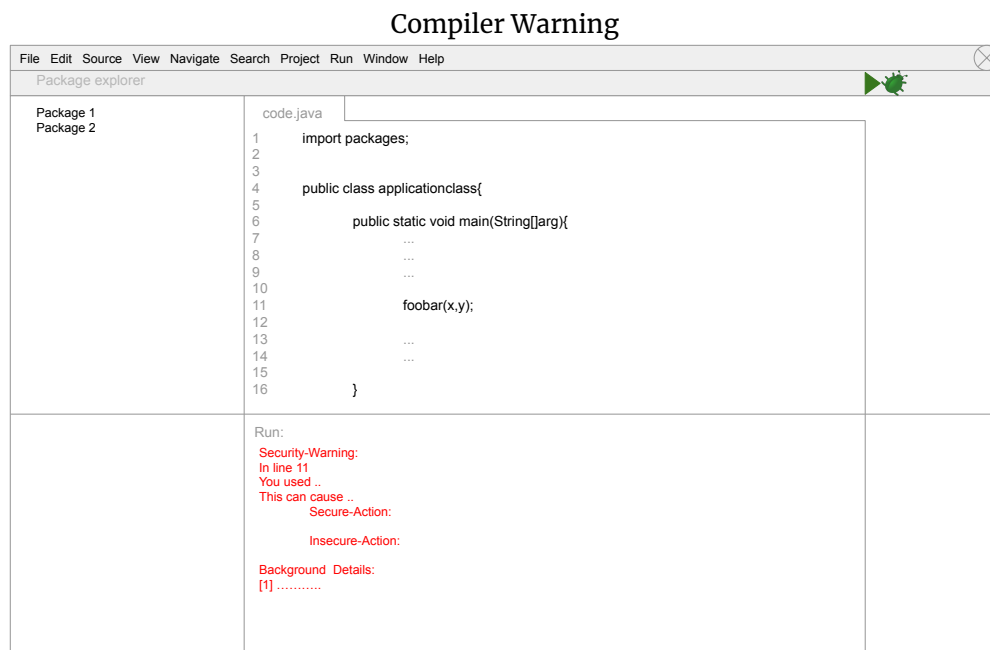


Figure A.15.: Compiler Warning
The compiler warning is displayed in the command line output of the IDE.

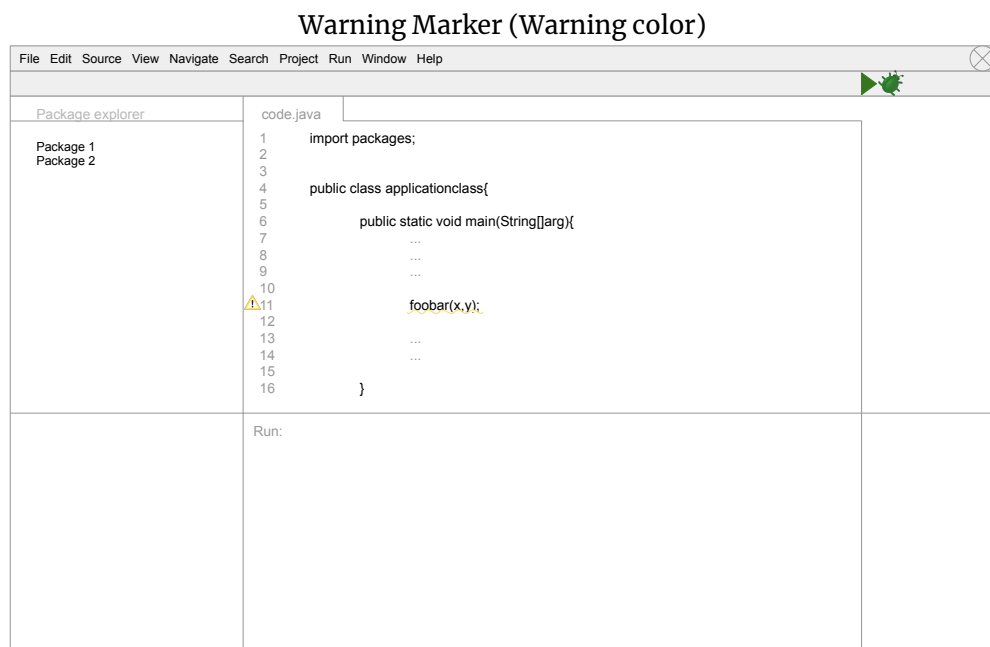


Figure A.16.: Warning Marker Yellow
The yellow marker is displayed in the code editor marking the exact line.

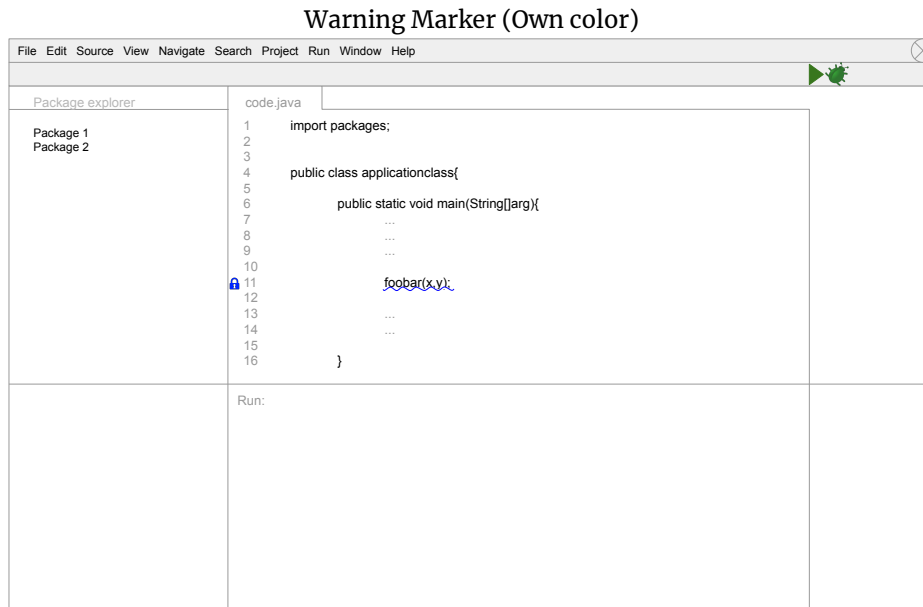


Figure A.17.: Warning Marker Own Color
The blue marker is displayed in the code editor marking the exact line.

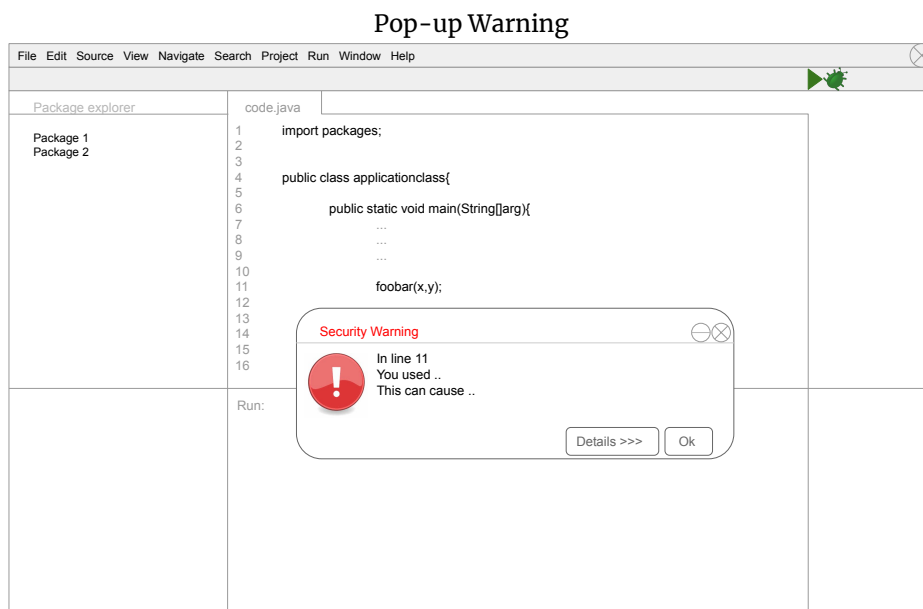


Figure A.18.: Pop-up Warning
The pop-up overlays the IDE and code editor.

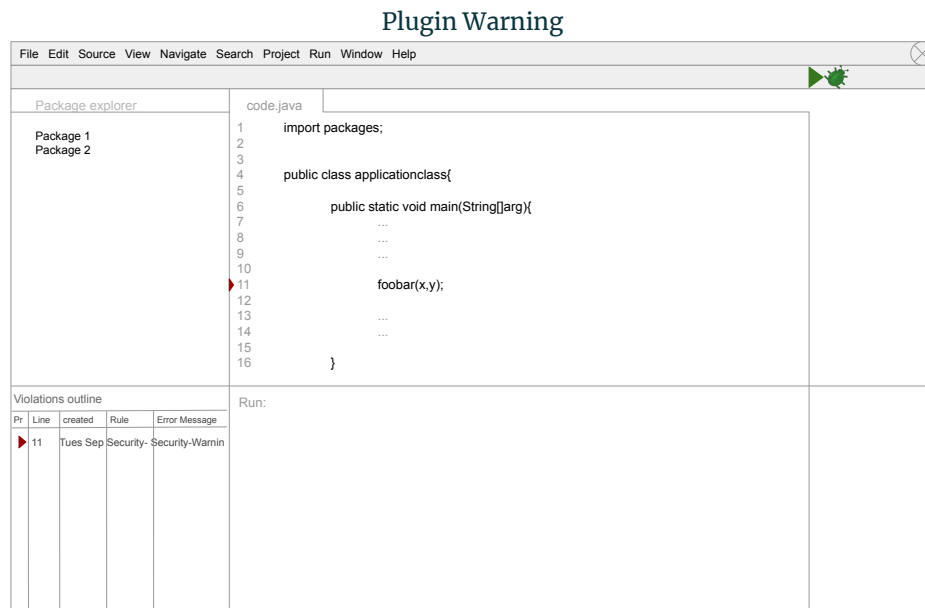


Figure A.19.: Plugin Warning

The plugin warning is displayed, by default the view can be found in the left corner of the IDE. It usually contains different other warnings than security warnings.

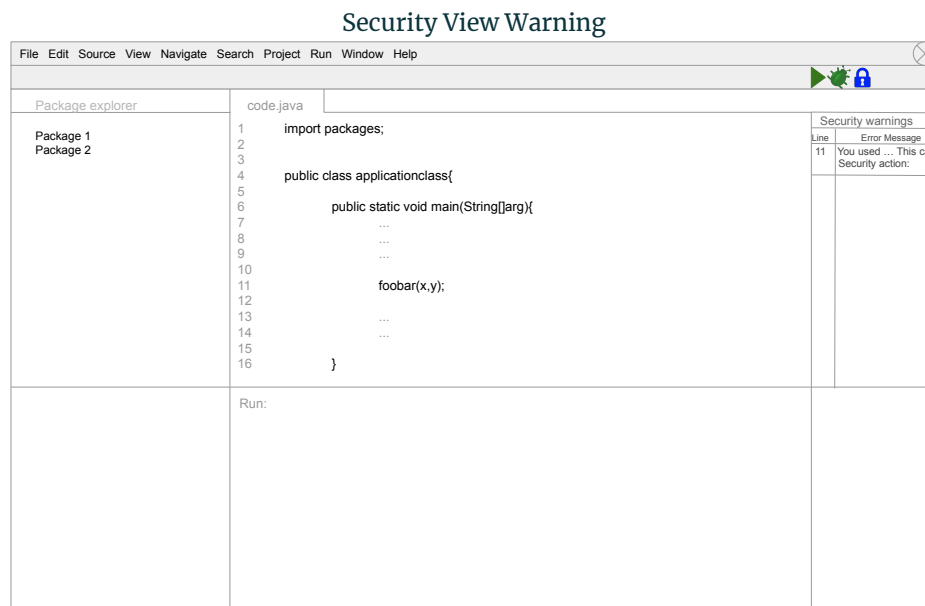


Figure A.20.: Security View

The security view warning is displayed in a view, by default the view can be found in the right corner of the IDE. Only security warnings would be displayed in the security view.

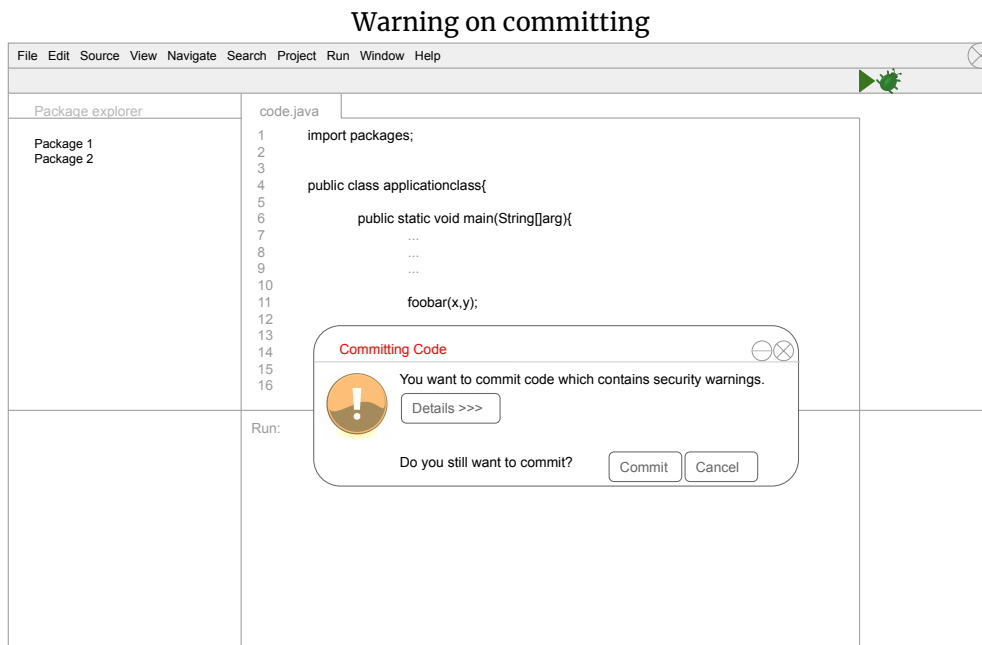


Figure A.21.: Warning on Committing
The warning is displayed on committing.

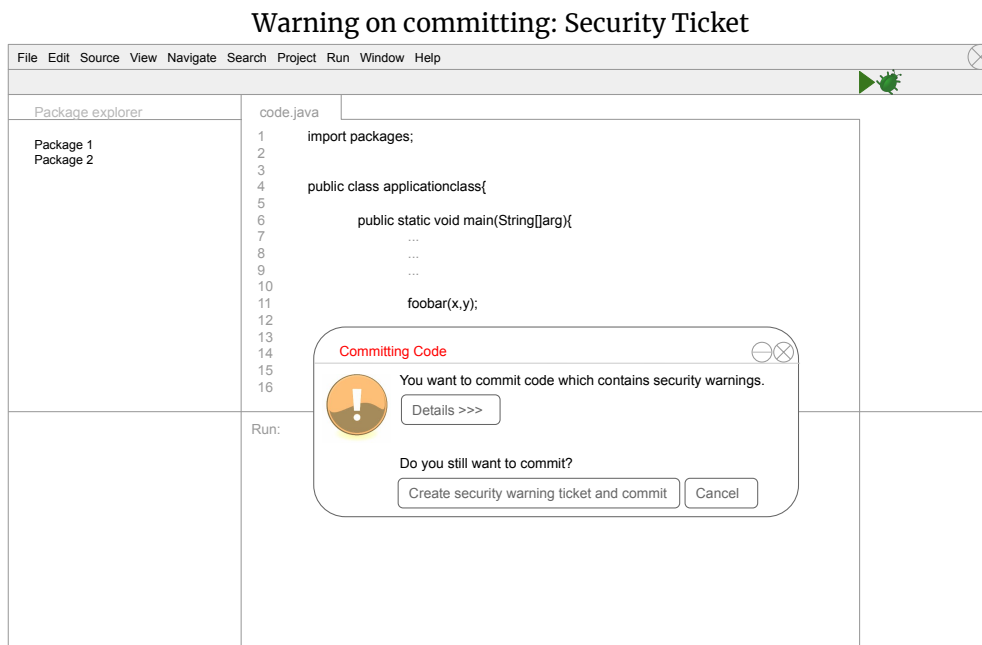


Figure A.22.: Warning on Committing with Automatic Security Ticket
The warning is displayed on committing. It is only possible to commit when creating a security ticket.

Example	Category	Sub-Category	Codes
<p>"That's why I like the idea of security view, it's kind of like plug-in, but you can differentiate it, we can see the plug-ins outline somewhere else and the security view warning somewhere else." (D5)</p>	Warning Preferences	Security View Warning	likes the idea of extra view for security
<p>"I didn't like it that much, because I think it's not that visible, visibility purpose like in the corner, because there is lots of other stuff." (S6)</p>			Security View Warning not visible
		<p>Compiler Warning</p> <p>Warning on Committing</p>	<p>Security View Warning: likes error with line number very interesting when using many tools summary in extra window - would be ok, too Security View Warning good overview can also be overkill bad: requires familiarization Security View Warning for severe security mistakes Security View Warning for HTTP instead of HTTPS Security View Warning for Static IV warning case Security View Warning could be combined with Warning Marker make sure that window is noticed whether there is enough space in this IDE</p> <p>likes Compiler Warnings Security Warnings as errors Compiler Warnings more familiar Compiler Warning good because of visibility Compiler Warning as favorite misleading could be overseen developers could also use command line dislike: it's more like an error that we get in coding Compiler Warnings could be not shown if code is not compiled Compiler Warning could be overseen in big projects always depends on how much scope I have Compiler Warning during coding would like to see Compiler Warning every time I compile code the good thing about Compiler Warnings is that you can disable them Compiler Warning for HTTP(S) Compiler Warning for SQL input validation</p> <p>prefers Warning on Committing</p> <p>likes Warnings on Committing Warning on Committing is a good idea because of visibility</p>

			<p>reminds me when I want to disable this Warning for testing but I like to have another stop sign THEN Warning on Committing necessary somebody already thought about it Committing Warning good idea after finished functionality task</p> <p>does not fit into my workflow always commit my code using the command line Warning on Committing for severe security issues allowance to commit only after solved security issues</p> <p>Warning on Committing every time others should see the security issue as well on merge request would make more sense when pushing into the master branch</p> <p>likes the warning project management doesn't want to have unsolved todos go into some pot into which nobody takes a look I reported it, so I don't have to do anything about it anymore! good for security process, not for development in general would make security administrator happy good when I can disable it depends on how experienced your co-workers are</p> <p>yellow: could be overseen: another color than yellow best: prefers Warning Marker in own color like icon for security I like this lock, this triangle open lock a special one, for example this icon like Warning Marker visible information directly with source code Warning Marker good during programming Warning Marker demonstrates security issues well Warning Marker useful inline is difficult if you have 10.000 lines of code own experience with Warning Markers negative strange point in time and I cannot believe it anyway would ignore Warning Marker in yellow Warning Marker meaningless Warning Marker during programming instead of Test Mode Warning Marker as addition</p> <p>it is unnecessary that it is always there likes the idea of having a list prefers Plugin Warning Plugin Warning reliable dislike: we already have lots of plugins in Eclipse it is a separate thing and therefore less important Plugin Warning not visible</p>
		Security Ticket	
		Warning Marker	
		Plugin Warning	

Pop-up Warning	<p>window for Plugin Warning could be hidden (not recommended) with a plugin option you would have so many errors or warnings bad: requires familiarization</p>
Own Warn- ing Design	<p>for me "Are you sure" questions are allowed to appear as a Pop-up Pop-up fine if requested for more details Pop-ups only in context Pop-up on committing Pop-up Warning for important security issues so if you have a Pop-up, you're forced to check all of them just at the end Pop-up bad because jump, a link not enough space to show error report</p> <p>Pop-up warning intrusive particularly when one uses shorthand symbol not when saving, not when editing with keyboard and you don't want to always have a window open</p> <p>Pop-up Warnings intrusive during coding blocker habituation effect but no Pop-up, I will simply close it depend on frequencies depends on development lifecycle time functionality first, Pop-up Warning second Pop-up Warning when it occurs and after finished coding</p> <p>have statistics for coding possibility to get details to security issue where to get additional knowledge to topic description and maybe additional help for some positions a more detailed description helpful forum build infrastructure where you can get feedback from more people to-do in code, automatically create a view out of it profile of preferences, suggest those text completion (parameter choices) for security warnings should show context issues, not only single lines above visual warnings or light goes out Warning as audio? Smart watch with electric shocks internet disabled for 10 minutes or so likes suggestions for improvements Warnings have to be short and concise Warnings should be short with possibility for more details Warning as drop-down text possibility for sorting</p>

		<p>Color for Warnings</p> <p>yellow Security warnings deserve an own color yellow could be overseen blue for Warnings red for Warnings color similar to red orange lilac for warnings color for warnings should depend on how important (security) is background color own color plus own symbol Warning sign more significant than color</p> <p>Combination</p> <p>combination of warnings is a good idea Compiler Warning + Warning Marker Compiler + Committing Compiler + Committing + Marker Warning Compiler + Plugin Warning Committing + Pop-up Committing + Warning Marker Committing + Warning Marker (own color for security/symbol) Committing + Warning Marker + Security View committing + Security View Warning Marker + own color + Plugin Warning Marker + Pop-up Warning Warning Marker own color + Security View Warning does not like the idea of combined warnings</p>
<p>"You have a testing environment for which I am currently implementing new features. In this case I would say that warnings can be clicked away." (D4)</p>	<p>Disabling warnings</p>	<p>ignore in test cases should be possible</p> <p>dismiss until next commit Reminder! after ignoring should be marked Test-Mode should be caught if forgotten Configuration! whether this is useful in the intended purpose administration costs! for security unavoidable ignore it for the run</p> <p>likes the idea of the possibility to ignore warnings for production</p> <p>disabling warning function usefulness depends on situation Warnings should always be shown warning disable function is not useful for security issues via SVN/Git solvable, e.g. Brunch</p>
<p>"Compiler warnings are not intrusive enough." (D3)</p>	<p>Warning in the context</p>	<p>important security issues should be warned intrusively</p> <p>better education instead of warnings!</p>

			<p>current warning system prevents from developing and does not help More than one perfect system Depends on habits, experience and different views Warnings important in java context or wherever in the outer world Warnings should be always up-to-date for you! APIs are only as secure as the developer warnings should be obligatory for security I would force the developer to actually solve the issue these warning cases easy to prevent really good, if you can get the warnings for basic steps Asks for static analyzer never seen a security warning in the IDE warnings for secure password storage useful warning appears directly after pulling IntelliJ pretty good for development quality control to include a last obstacle</p>
"We do have a security team in our company but they are not familiar with coding. In the end they are only responsible for imposing security instructions on us." (D13)	Company		security team in company
"There is a security officer. But I could not directly approach him, he is over-strained because he needs to make sure that we comply with security policies. There is a security guideline but it is not precisely made for all individual cases. So I cannot look up and see which code I have to use. That is what the developer decides on his own." (D9)			security officer available, but cannot be asked
			<p>superordinated security check impose security guideline are not familiar with coding security responsible person decides about security issues for inspector-tool if somebody is responsible and delegates no security responsible person Code Review I think this should be done for all projects. I am also one of the initiators of this code review of course doubled resources Testing! no demand for security in company, but policy/guideline general policy/guide, no specific guideline for current project security responsible person not relevant for small team</p>
"If I am working with a productive team where everybody works fine, then they would look at it. But usually, there are different people in a team. The people working system-oriented would look at it closely. The front-end developer would not likely do that."(D14)	Team		<p>colleagues with skills could check the issues one more time</p> <p>Warnings from other people</p>

			<p>would like to see warnings of other people maintainer responsible for security</p> <p>I reported it, so I don't have to do anything about it anymore!</p> <p>to-dos are often not comprehensible after a while</p> <p>depends on how the team is organized it is nothing which is easy to realize depends on the team size relatively small set it is difficult during vacation depends on how frequently colleagues change would fix others warnings if working on same program section usually developer develops alone delegating warnings to colleagues with other context no sense</p>
"Of course. Also, we are a government entity. It is very important for us and we get reports and information sources regularly telling us which rules we need to follow. Within the scope of a project me as a project manager has to make sure that the security policies are met." (D6)	Responsibility for Security		<p>feel responsible for security</p> <p>if you program something, you are responsible for it personal preference that I additionally get pmd's opinion consider security while programming project management government agencies have security guidelines ok this is not my code could use frameworks which ensure security if there is a mistake in it, I don't feel ... of course people developing security guidelines not technique there exists the owasp-guideline and this needs to be fulfilled developer overloaded</p>
"Also I have some negative experience because it hinders (laugh). When we program they need to release the code for production or the testing environment, there is a programming running checking for specific criteria. Some criteria is too specific. It does not fit everything in the code and that hinders you a bit." (D12)	Security		<p>security checks prevent from working</p> <p>extra warnings for security security warnings for developers important security should be there by default often security is only used, because no extra steps needed relies on tools which deal with security</p>
"Personally, I am not affected, but if you work for a company you hear some things. For example if there are some updates we need to make sure the systems are up-to-date and security vulnerabilities are always patched."(D6)	Security Breach		<p>did not experience security breach as developer</p> <p>experienced security breach as end-user</p>

			used honeypots to find vulnerabilities fetched scripts security breaches are in the php context right now firewall system word macro direct access to SAP database security updates ignored certificate security as developer already
"I would like to see directly if the IDE thinks there is a problem anywhere." (D2)	Time		Warnings during programming best immediate prompt when IDE sees problem during and after finished programming no time to work to learn new tools Depends on time, not on warning type! warnings could be too late So it's really good, if you come to know before committing 2-3 times a day at most depends on how severe security issues are warnings during programming and on committing education time security trade off because in between.. I think it is disturbing then at the end warnings need extra time you need a lot of time for security quality costs time
"I mean too many warnings are annoying but being able to check it makes sense." (D2)	Frequency		too many warnings are annoying warnings everywhere (habitation)
"Yes. The standard rules. That is the case. You see a lot of warnings most of them get deactivated because they are not applicable." (D10)	Used tools		pmd: many warnings, most is deactivated, not applicable sonarqube: I thought it was interesting sonarqube: feedback via pop-up FindBugs Virtual Forge SAP-Code Inspector guideline and then you can derive things pmd: if you load everything, they sometimes disagree with each other pmd: default settings are not really helpful pmd: we would use it afterwards
"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)	Functionality vs. security		functionality first, security second
"In project I worked on it is not really checked. That is my personal preference that I would like to hear the opinion of pmd." (D10)			on the projects I am working on, it is not really checked
"You need a lot of time for security. But when you have time pressure everything needs to be done fast so then how much more time would you like to invest?" (D7)			time pressure, security needs time

"When you are developing for a bank you internalized it (security)." (D12)			functionality and security equally important
"If there are security relevant findings in the code, they are displayed and if they are critical, the code is not released." (D13)			if critical, code is not released
"Checking code and reviewing code is not a pleasant task. That's how it is. But it is required. Therefore, I would say merge should not work [if there is a warning]. Then you need to really look why it does not work." (D4)			no merge request allowed if security issues are not fixed
"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)			functionality, security trade off
			distracts a lot from what you actually want to code Errors first, warnings second yes that come while conceptualizing security does not matter often for us other things are more important... performance

Table A.1.: Codebook

B. Appendix: Data Quality: Screening Questions for Online Surveys with Programmers

B.1. Structure

The appendix is structured as follows.

1. **Section B.1** outlines the structure of the appendix.
2. In **Section B.2**, we provide information on our survey questions.
3. In **Section B.3** we give details on the demographics of our participants.
4. **Section B.4** summarizes results on the effectiveness and efficiency of our tested screener questions. Additionally, this section also shows an overview of information resources our participants from the attack scenario used to answer the programming questions.
5. **Section B.5** reports the timing thresholds for the 4 screener questions Q2, Q3, Q4, and Q6, which we recommended to use with time limits.

B.2. Survey

The correct answers are marked with . All questions were randomly shown to the participants.

Q1.1 Which of these programming languages have you worked with before?

- C#
- C
- C++
- Python
- JavaScript
- Java
- Ruby
- PHP
- Shell
- TypeScript
- Other:
- I don't program

Q1.2 Which of these lesser-known programming languages have you worked with before?

- Yod
- Lore
- Torg
- LPrime
- ThreeP
- EMH
- Holly
- SHROUD
- LTCdata
- Kryten
- None of the above

Q2 Which of these websites do you most frequently use as aid when programming?

- Wikipedia
- LinkedIn
- Stack Overflow
- MemoryAlpha
- I have not used any of the websites above for programming
- I don't program

Q3 Choose the answer that best fits the description of a compiler's function.

- Refactoring code
- Connecting to the network
- Aggregating user data
- I don't know
- Translating code into executable instructions
- Collecting user data

Q4 Choose the answer that best fits the definition of a recursive function.

- I don't know
- A function that runs for an infinite time
- A function that does not have a return value
- A function that can be called from other functions
- A function that calls itself
- A function that does not require any inputs

Q5 Choose the answer that best fits the description of an algorithm:

- A set of instructions that, when executed, create a desired result
- I don't know
- Data that is required to run a program
- A software tool for data analysis
- The inverse of an exponential function
- A program that is run in the background on websites

Q6 Which of these values would be the most fitting for a Boolean?

- Small
- I don't know
- Solid
- Quadratic
- Red
- True

Q7 Please pick all powers of 2:

- 218
 - 16
 - I don't know
 - 86
 - 512
 - 100
-

—
Q8 Please translate the following binary number into a decimal number **101**:

- 5
- 9
- 7
- I don't know
- 4
- 3

Q9 Please select all even binary numbers:

- I don't know
- 100101
- 10010110
- 101101
- 11010
- 11011011

Q10 Please select all valid hexadecimal numbers:

- FACE
- HEAD
- 1234
- 99FF
- 78GH
- I don't know

Q11 When multiplying two large numbers, your program unexpectedly returns a negative number. What might have caused this?

- Backflow
- Interflow
- Overflow
- Controllflow
- I don't know
- Underflow

Q12 What is the run time of the following code?

```
for 0 to n
  for 0 to n
    foo()
```

- I don't know
 - Logarithmic
 - Cubic
 - Linear
 - Quadratic
 - Constant
-

Given the following pseudocode algorithm:

```
1: arr = [1,4,6,7,9,2,3,5,8,0]
2: i = 1
3: j = 0
4: while i <= 9:
5:   while j <=9:
6:     if arr[j] > arr[j+1]
7:       swap( arr[j], arr[j+1])
8:     j++
9:   j = 0
10: i++
```

Q13 When running the code, you get an error message for line 6: Array index out of range.

What would you change to fix the problem?

- Line 6: if arr[j] > arr[j-1]
- Line 9: j = i
- Line 2: i = 0
- I don't know
- Line 5: while j < 9:

Q14 What is the purpose of the algorithm?

- Sorting the array
 - Count the total number of items in the array
 - Count the unique items in the array
 - I don't know
 - Selecting a number random from the array
-

```
main{
  print(func("hello world"))
}

String func(String in){
  int x = len(in)
  String out = ""
  for(int i = x-1; i >= 0; i--){
    out.append(in[i])
  }
  return out
}
```

Q15 What is the parameter of the function?

- String out
- String in
- I don't know
- int i = x-1; i >= 0; i--
- Outputting a String
- int x = len(in)

Q16 Please select the returned value of the pseudocode above:

- hello world
- hello world 10
- dlrow olleh
- world hello
- HELLO WORLD
- I don't know
- hello world hello world hello world hello world

1. **[Attention Check]** This is an attention check question. Please select the answer "Octal".

- Duodecimal
- I don't know
- Octal
- Binary
- Decimal
- Hexadecimal

Demographic Questions

1. How many years of programming experience do you have?
 2. How experienced would you consider yourself at programming?
 - Beginner
 - Intermediate
 - Expert
 - No experience at all
 3. Have you ever been paid for your work as a programmer?
 - Yes
 - No
 4. In which country do you currently reside?
-

5. How old are you?
6. What is your gender?
 - Male
 - Female
 - Prefer to self-describe:
 - Prefer not to tell
7. What is your main profession?

B.3. Participants' Demographics

B.3.1. Country of Residence

Clickworker No Experience:

NA: (1), Finland (1), France (1), Germany (21), India (3), Italy (1), Netherlands (1), Philippines (1), Russian Federation (1), Serbia (1), South Africa (2), Spain (2), United Kingdom of Great Britain and Northern Ireland (8), United States of America (6)

Clickworker with Programming Experience:

Argentina (1), Austria (1), Egypt (2), Finland (1), Germany (22), Greece (1), India (4), Indonesia (1), Italy (4), Kenya (1), Netherlands (1), Nigeria (1), Portugal (1), Romania (1) Spain (1), Sweden (1), United Kingdom of Great Britain and Northern Ireland (5), United States of America (3)

Clickworker Attack:

Australia (1), Austria (1), Brazil (3), Colombia (1), Finland (1), France (1), Germany (16), India (1), Italy (4), Kenya (2), Malaysia (1), Mexico (1), Peru (1), Poland (2), Romania (1), Spain (4), Sweden (1), United Kingdom of Great Britain and Northern Ireland (1), United States of America (4)

B.3.2. Main Occupation

Professional Developers:

Lead Developer (3), Software Developer (22), IT Staff (2), Data scientist (1), Engineer (1), IT Release Manager (1), System architect (1), Function Developer (1), Security Consultant (1)

Clickworker without Programming Skill:

Student (4), House wife (3), Freelancer (3), Engineer (3), Merchant (2), Teacher (2), Administrator (2), Developer (2), Editor (2), Social worker (1), Self-employed (1), Manager (1), Druggist (1), Scientist (1), Project manager (1), Telephone Operator (1), Security (1), Office clerk (1), Public administration specialist (1), Bookkeeping (1), Paralegal (1), Investment analysis (1), Doctor (1), Unemployed (1), Physicist (1), Translator (1), Geologist (1), Tattoo artist (1), IT Security (1), Economist (1), Environmental Health Officer (1), Controller (1), Service staff (1), Support (1), Finance (1), NA (1)

Clickworker with Programming Skill:

Student (9), Software Developer (8), Support (7), IT (4), Self-employed (2), House wife/husband (2), Law (1), Sales (1), Spiritual life coaching (1), Nurse (1), Data scientist (1), Office employee (1), Engineer (1), System administrator (2), Scientist (4), Chemist (1), Assistant lecturer (1), Unemployed (1), Graphic designer (1), Architecture (1), Astronomer (1), Nutritionist (1)

Clickworker Attack:

Student(4), Clerk (3), Sales (3), Manager (2), Teacher (2), Worker (2), Administrative (1), Architecture (1), Asacom (1), Business (1), Carrier (1), Civil engineering (1), Crowdworker (1), Data handling (1), Data scientist (1), Delivery driver (1), Developer (1), Factory technician (1), Finance (1), Freelancer (1), IT (1), Landscape architecture (1), Editor (1), Material technician (1), Mechanic (1), Medical doctor (1), Media (1), Musician (1), Online shop operator (1), Physiotherapist (1), Programming (1), Independent (1), Software developer (1), Social pedagogue (1), Student developer (1), Student assistant (1), NA (1)

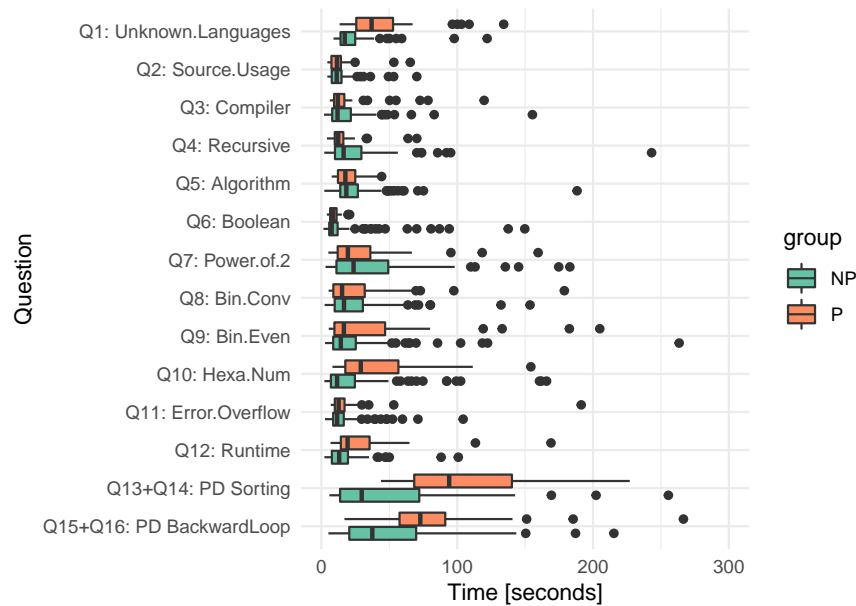


Figure B.1.: Time to answer each question for the non-programmer (NP) and programmer (P) group.

B.4. Results

B.4.1. Effectiveness

Table B.1 shows an overview of the correct and incorrect answers of the programmer and non-programmer group for all the questions (Q1-Q16). Table B.3 summarizes the related statistical analysis results. Figure B.2 displays the number of correct solutions of the non-programmer group ($n = 100$) separated by participants who indicated to have 0 years and more than 0 years of programming experience.

B.4.2. Efficiency

We visualized the mean times for both programmer and non-programmer groups according to each task block in Figure B.1.

B.4.3. Attack Scenario

Table B.4 shows an overview of the self-reported resources used by participants for correctly answering the questions within the attack scenario ($n = 47$).

B.5. Timing thresholds

Figures B.3, B.4, B.5, and B.6, illustrate how many participants from the attacker (attack) and programmer group (progs) solved the four recommended questions Q2, Q3, Q4, and Q6 correctly and how many seconds they needed to answer. Drawing a line on a certain time threshold, it becomes visible how many attackers and programmers would be excluded from the correct group.

Table B.1.: Number of (in)correct answers of the programmers (n = 50) and non-programmers (n = 100) for Q1-Q16.

	✓	✗		✓	✗ (Idp)		✓	✗ (Idk)
Programmer	49	1	Programmer	50	0 (0)	Programmer	50	0 (0)
Non-Programmer	91	9	Non-Programmer	6	94 (60)	Non-Programmer	33	67 (37)
(a) Unknown.Languages (Q1)			(b) Source.Usage (Q2)			(c) Compiler (Q3)		
	✓	✗ (Idk)		✓	✗ (Idk)		✓	✗ (Idk)
Programmer	50	0 (0)	Programmer	49	1 (0)	Programmer	50	0 (0)
Non-Programmer	30	70 (33)	Non-Programmer	47	53 (13)	Non-Programmer	25	75 (60)
(d) Recursive (Q4)			(e) Algorithm (Q5)			(f) Boolean (Q6)		
	✓	✗ (Idk)		✓	✗ (Idk)		✓	✗ (Idk)
Programmer	48	2 (0)	Programmer	48	2 (0)	Programmer	48	2 (0)
Non-Programmer	41	59 (18)	Non-Programmer	38	62 (38)	Non-Programmer	38	62 (38)
(g) Power.of.2 (Q7)			(h) Bin.Conv (Q8)					
	✓	✗ (Idk)		✓	✗ (Idk)		✓	✗ (Idk)
Programmer	45	5 (2)	Programmer	35	15 (0)	Programmer	35	15 (0)
Non-Programmer	34	66 (39)	Non-Programmer	6	94 (51)	Non-Programmer	6	94 (51)
(i) Bin.Even (Q9)			(j) Hexa.Num (Q10)					
	✓	✗ (Idk)		✓	✗ (Idk)		✓	✗ (Idk)
Programmer	47	3 (2)	Programmer	40	10 (5)	Programmer	40	10 (5)
Non-Programmer	21	79 (49)	Non-Programmer	6	94 (56)	Non-Programmer	6	94 (56)
(k) Error.Overflow (Q11)			(l) Runtime (Q12)					
	✓	✗ (Idk)		✓	✗ (Idk)		✓	✗ (Idk)
Programmer	38	12 (2)	Programmer	49	1 (0)	Programmer	49	1 (0)
Non-Programmer	9	91 (66)	Non-Programmer	24	76 (51)	Non-Programmer	24	76 (51)
(m) Error.OutOfBound (Q13)			(n) Sorting.Array (Q14)					
	✓	✗ (Idk)		✓	✗ (Idk)		✓	✗ (Idk)
Programmer	50	0 (0)	Programmer	47	3 (0)	Programmer	47	3 (0)
Non-Programmer	13	87 (57)	Non-Programmer	7	93 (41)	Non-Programmer	7	93 (41)
(o) Function.Param (Q15)			(p) Backward.Loop (Q16)					

Idp = I don't program; Idk = I don't know.

Question	O.R.	CI	p-value
Q1	4.8	[0.63, 216.47]	0.17
Q2	Inf	[143.54, Inf]	< 0.001*
Q3	Inf	[23.93, Inf]	< 0.001*
Q4	Inf	[27.37, Inf]	< 0.001*
Q5	54.18	[8.56, 2240.58]	< 0.001*
Q6	Inf	[34.71, Inf]	< 0.001*
Q7	33.82	[8.09, 302.74]	< 0.001*
Q8	38.29	[9.14, 343.35]	< 0.001*
Q9	17.11	[6.05, 60.28]	< 0.001*
Q10	35.14	[12.11, 120.60]	< 0.001*
Q11	56.86	[15.96, 310.37]	< 0.001*
Q12	59.35	[19.37, 218.09]	< 0.001*
Q13	30.79	[11.45, 92.27]	< 0.001*
Q14	149.39	[23.1, 6075.50]	< 0.001*
Q15	Inf	[71.76, Inf]	< 0.001*
Q16	190.22	[45.97, 1170.64]	< 0.001*

Table B.3.: Summary of statistical analysis for questions Q1-Q16 for the programmer and non-programmer group.

Fisher's exact tests were used for analysis. The independent variable was the programmer/non-programmer group. The dependent variable was the correctness of an answer. Significant results are marked with *.

Question	No of correct responses	Used Internet search	Asked friends/colleagues	Other
Q2	29	12	4	0
Q3	36	14	4	1: "I actually knew that myself! :)"
Q4	36	19	3	0
Q6	36	16	2	1: "I knew it could have a value of true or false"
Q14	29	0	1	1: "Guessed"
Q15	13	0	0	0
Q16	12	1	2	0

Table B.4.: Self-reported resources used by participants for correctly answering the questions within the attack scenario (n = 47). Multiple answers were possible.

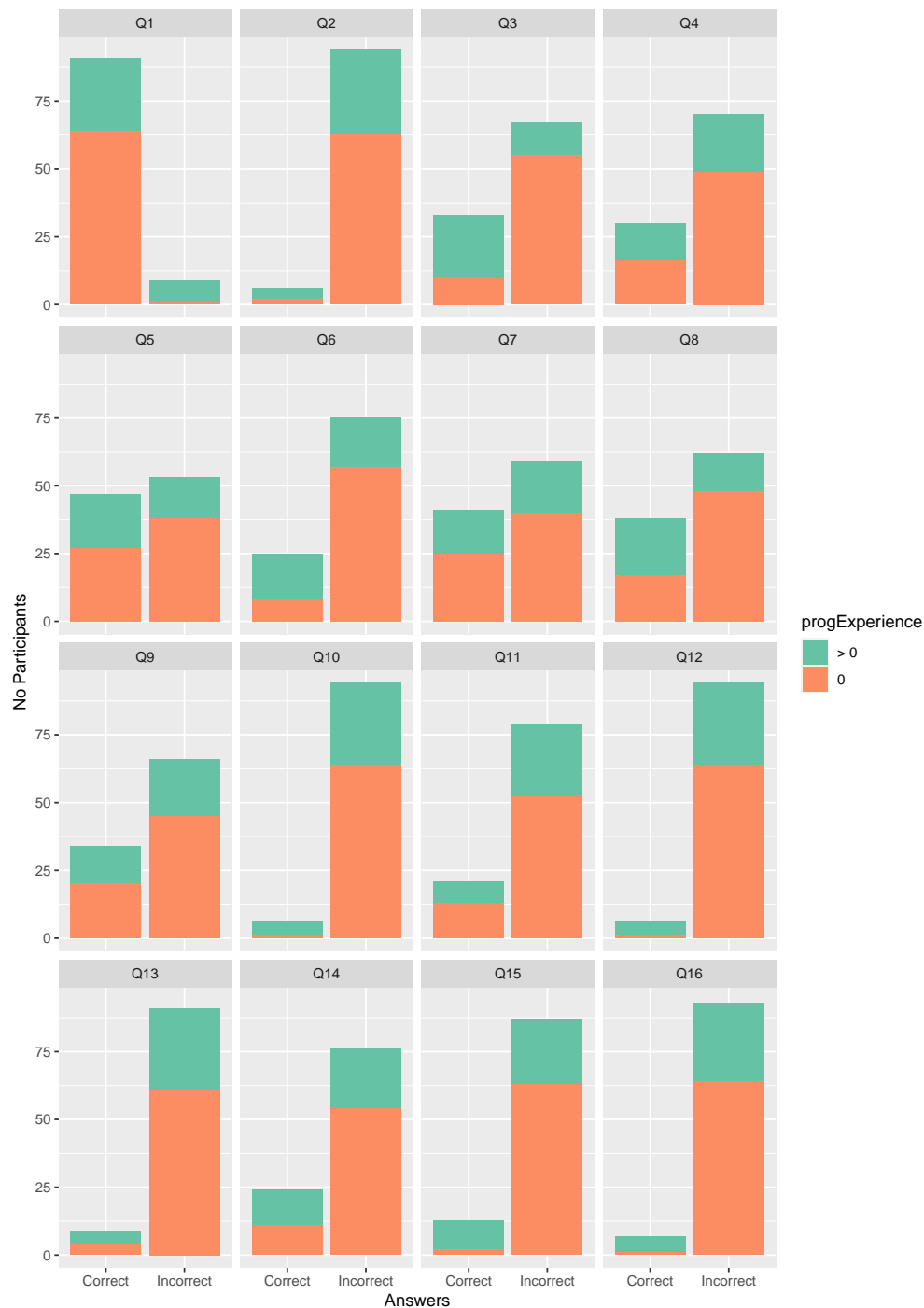


Figure B.2.: Number of correct solutions of the non-programmer group ($n = 100$) separated by participants who indicated to have 0 years and more than 0 years of programming experience.

progExperience: > 0 if non-programmer participants reported to have more than 0 years of programming experience, and $= 0$ if they reported to have no programming experience at all.

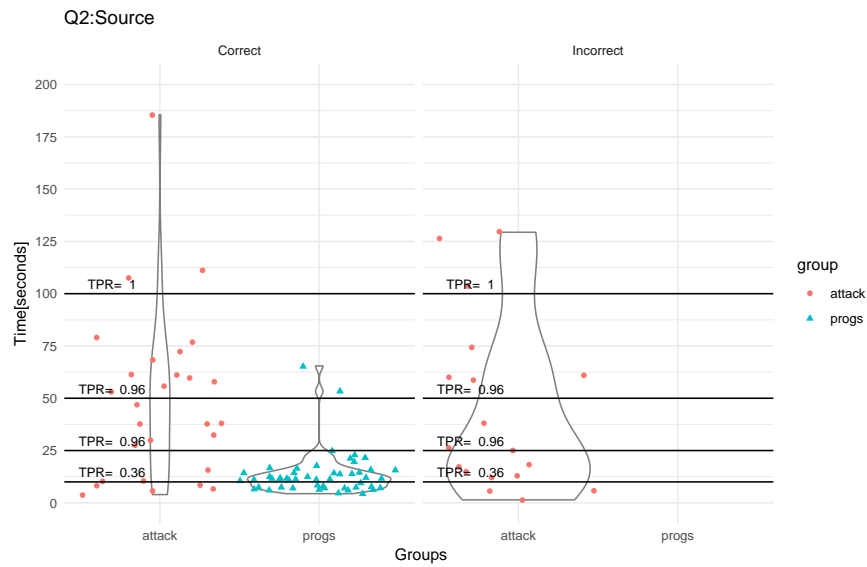


Figure B.3.

The figure illustrates the true positive rate at example thresholds (10 seconds, 25 seconds, 50 seconds, 100 seconds)

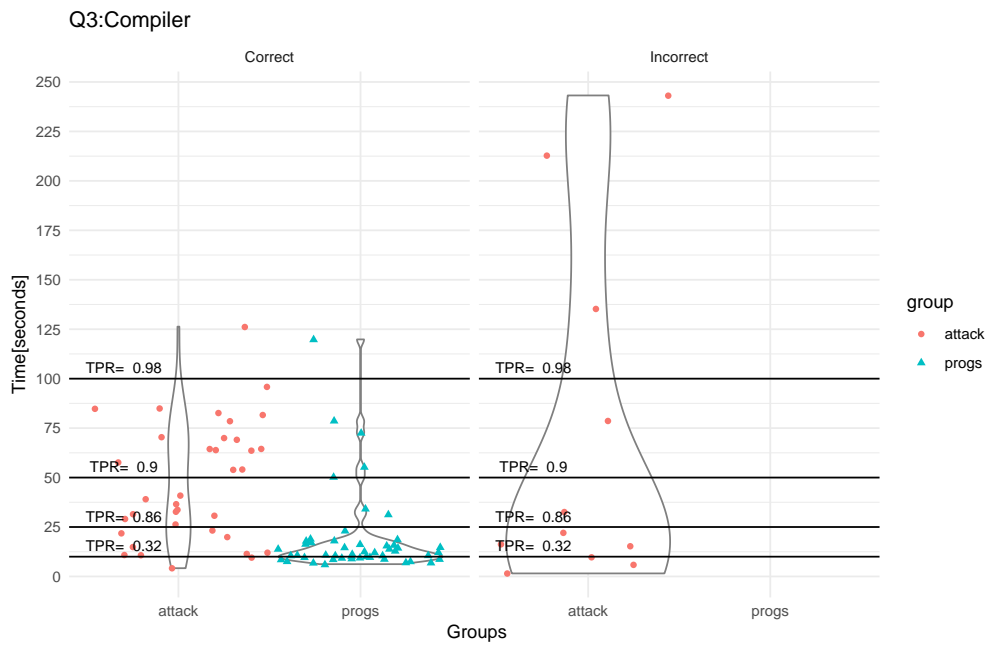


Figure B.4.

The figure illustrates the true positive rate at example thresholds (10 seconds, 25 seconds, 50 seconds, 100 seconds).

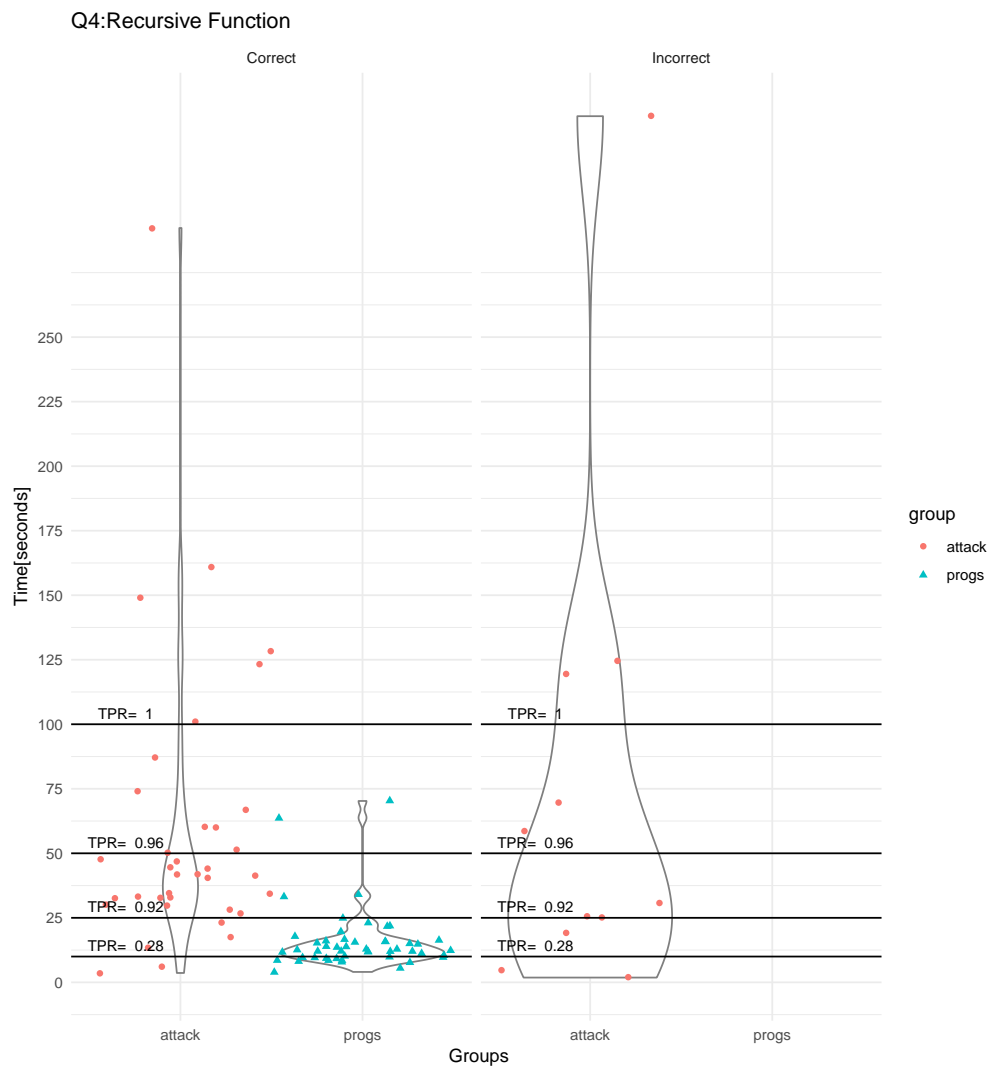


Figure B.5.

The figure illustrates the true positive rate at example thresholds (10 seconds, 25 seconds, 50 seconds, 100 seconds)

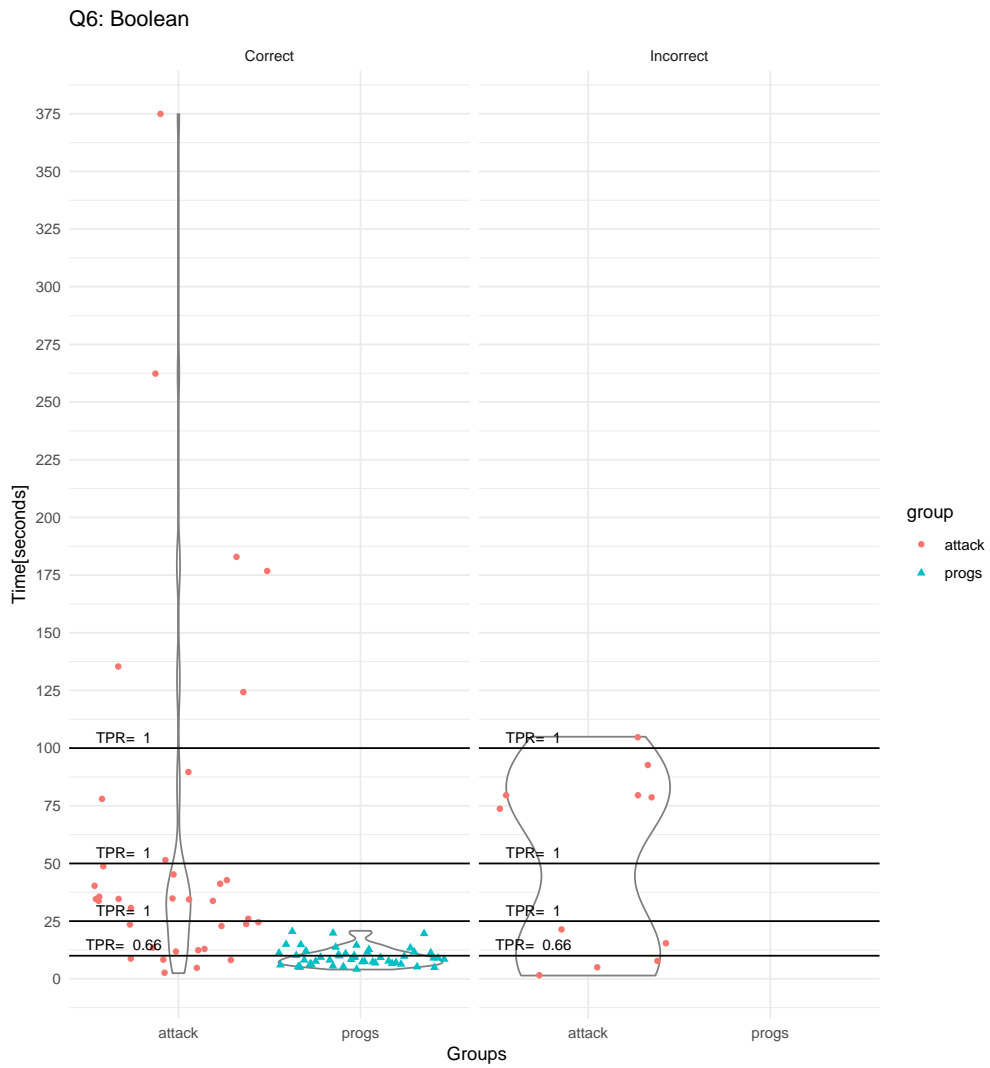


Figure B.6.

The figure illustrates the true positive rate at example thresholds(10 seconds, 25 seconds, 50 seconds, 100 seconds)

C. Appendix: Data Quality: Time Limits in Screener Questions

C.1. Questions

C.1.1. Attention Check Question

This is an attention check question. Please select the answer "Octal".

- Octal
- Decimal
- Hexadecimal
- Duodecimal
- I don't know

C.1.2. Demographic Questions

1. Which of these programming languages have you worked with before?

- Python
- Java
- PHP
- C#
- C++
- Shell
- TypeScript
- Ruby
- C
- Other
- None

2. How many years of programming experience do you have?

3. How experienced would you consider yourself at programming?

- Beginner
- Intermediate
- Expert
- No experience at all

4. Have you ever been paid for your work as a programmer?

- Yes
- No

5. In which country do you currently reside?

6. How old are you?

7. What is your gender?

- Male
- Female
- Prefer to self-describe:
- Prefer not to tell

8. What is your main profession?

C.1.3. Screener Questions

In the following, all tasks used in this study are presented in alphabetical order. The correct answers are marked and the potential time limit is noted. Answers were presented to the participants in randomized order for each task.

Task: Array, Time limit: 60 seconds

Given the array `arr[7,3,5,1,9]`, what could the command `arr[3]` return? The array starts with 0.

- 1

- 7, 3, 5
- 21, 9, 15, 3, 27
- 3
- 10, 6, 8, 4, 12

Task: Backward.Loop, Time limit: 220 seconds

```
1: main{
2:     print(func("hello world"))
3: }
4:
5: String func(String in){
6:     int x = len(in)
7:     String out = ""
8:     for(int i = x-1; i >= 0; i--){
9:         out.append(in[i])
10:    }
11:    return out
12: }
```

Please select the returned value of the pseudo code above. Arrays start with 0.

- dlrow olleh
- world hello
- hello world
- hello world hello world
- HELLO WORLD
- hello world hello world hello world hello world

Task: Boolean, Time limit: 30 seconds

Which of these values would be the most fitting for a Boolean?

- True
- Red
- Solid
- Quadratic
- Small

Task: Compiler, Time limit: 40 seconds

Choose the answer that best fits the description of a compiler's function.

- Translating code into executable instructions
- Connecting to the network
- Refactoring code
- Collecting user data
- Aggregating user data

Task: CountString, Time limit: 180 seconds

```
1: function text (string1, string2):
2:     length = min(string1.length, string2.length)
3:     counter=0
4:     for (int i = 0; i < length; i++):
5:         if (string1[i] == string2[i]):
6:             counter++;
7:     return(counter)
```

What is the purpose of the pseudo-code algorithm above? Arrays start with 0.

- Count number of the same characters at the same position for two strings
- Check if two strings are the same
- Check which string is the longest
- Count the letters in the larger string
- Count the letters in the smaller string

Task: Error.OutOfBounds, Time limit: 90 seconds

What issue would the following pseudo-code most likely cause? Arrays start with 0.

```
1: arr = [3,5,7,1,3]
2: tmp = arr[-1]+arr[2]
3: if(tmp>=0):
4:     print(arr[2])
5: else:
6:     print("tmp is smaller than 0")
```

- Index out of range
- Undefined behavior
- Syntax error
- Endless loop
- Type mismatch

Task: Error.Syntax, Time limit: 60 seconds

What issue would the following pseudo-code most likely cause?

```
1: string = "Hello World"
2: if (string.length%2 == 1):
3:     parity = "odd"
4: else:
5:     parity = "even"
6: print(parity)
```

- Syntax Error
- Syntax error
- Endless loop
- Type mismatch
- Undefined behavior

Task: IDE.Known, Time limit: 60 seconds

Please select 2 items from the list which are IDEs.

- Visual Studio
- Eclipse
- pyCharm
- gcc
- HTML
- CSS
- jpg
- C++
- UDP
- None of the above

Task: IDE.Used, Time limit: 30 seconds

Which of these IDEs have you used before?

- Visual Studio
- Eclipse
- pyCharm
- gcc
- HTML
- CSS
- jpg
- C++
- UDP
- None of the above

Task: Infinite.Loop, Time limit: 180 seconds

When executing the pseudo-code below, your program never stops running. What line would you add to fix the code? Arrays start with 0.

```
1: i = 0
2: while(i < array.length):
3:     value = array[i]
4:     j = i
```

```

5:     while (j>0 && array[j-1] < value):
6:         array[j] = array[j-1]
7:         j --
8:     array[j] = value
9: return(array)

```

- Between line 8 and 9: i++
- Replace line 4 with: if (i = j):
- Before line 1: j = 1
- Between line 6 and 7: i --
- Between line 1 and 2: value = 1

Task: Palindrome, Time limit: 120 seconds

```

1: function test(word[]):
2:     result = True
3:     for (i = 0, j = word.length - 1; i < word.length/2; i++, j--):
4:         if (word[i] != word[j]):
5:             result = False
6:             break
7:     return(result)

```

What is the purpose of the pseudo-code algorithm above? Arrays start with 0.

- Check if the input is spelt the same way backwards as forwards
- Check if inputs are equal
- Check if all letters of the input are unique
- Check if the number of letters is even

Task: Parameter, Time limit: 60 seconds

```

1: function countl(input):
2:     i = 0
3:     output = 0
4:     while i < input.length:
5:         if input[i] == 'l':
6:             output++
7:             i++
8:     return output

```

What is the parameter of the pseudo-code function? Arrays start with 0.

- input
- i < String.length
- output
- 0
- Outputting an integer

Task: Prime, Time limit: 180 seconds

```

1: function func(number):
2:     result = True
3:     for(int i = 2; i < number; i++):
4:         if(number % i == 0):
5:             result = False
6:             break
7:     return(result)

```

What is the purpose of the pseudo-code algorithm above?

- Check if a given number is prime
- Check if a given number contains the digit 0
- Check if a given number is equal to 0
- Find the smallest digit in a given number
- Check if i is smaller than the given number

Task: Recursive, Time limit: 30 seconds

Choose the answer that best fits the definition of a recursive function.

- A function that calls itself
- A function that does not have a return value
- A function that can be called from other functions
- A function that runs for an infinite time
- A function that does not require any inputs

Task: RecursiveFunction1, Time limit: 90 seconds

Is the following pseudo-code function a recursive function?

```
1: function reverse (array):
2:     array_reverse = []
3:     i = array.length - 1
4:     while i >= 0:
5:         array_reverse.append(array[i])
6:         i = i - 1
7:     return array_reverse
```

- No
- Yes

Task: RecursiveFunction2, Timelimit: 90 seconds

Is the following pseudo-code function a recursive function?

```
1: function even(int):
2:     if int == 1:
3:         return False
4:     elif int == 2:
5:         return True
6:     else:
7:         return even(int - 2)
```

- Yes
- No

Task: RecursiveFunction3, Timelimit: 90 seconds

Is the following pseudo-code function a recursive function?

```
1: function fac(i):
2:     if i == 1:
3:         return 1
4:     else:
5:         return i*fac(i-1)
```

- Yes
- No

Task: RecursiveFunction4, Timelimit: 90 seconds

Is the following pseudo-code function a recursive function?

```
1: function maximum(array):
2:     i = 0
3:     max = 0
4:     while i < array.length:
5:         if array[i] > max:
6:             max = array[i]
7:         i++
8:     return max
```

- No
- Yes

Task: Rec.Lang, Time limit: 30 seconds

Please select the two programming languages from the list below.

- Java
-

- C
- Kryten
- ThreeP
- Holly
- EMH
- LPrime
- Yod
- Torg
- None of the above

Task: Return, Time limit: 80 seconds

What would the pseudo-code return if i is 4?

```
1: function(integer i){
2:     if(i != i):
3:         return 0
4:     if(i == i):
5:         return 1
6:     if(i = i):
7:         return 2
8:     if(i + i):
9:         return 3
10:    return i
11: }
```

- 1
- 0
- 2
- 3
- 4

Task: Sorting.Array, Time limit: 240 seconds

Given the following pseudo-code algorithm:

```
1: arr = [1,4,6,7,9,2,3,5,8,0]
2: i = 1
3: j = 0
4: while i <= arr.length:
5:     while j < arr.length:
6:         if arr[j] > arr[j+1]
7:             swap(arr[j],arr[j+1])
8:             j++
9:         j = 0
10:    i++
```

What is the purpose of the algorithm? Arrays start with 0.

- Sorting the array
- Selecting a random number from the array
- Count the unique items in the array
- Count the total number of items in the array
- Reverse the order of the items in the array

Task: Source.Usage, Time limit: 30 seconds

Which of these websites is used most frequently by developers as aid for programming?

- Stack Overflow
 - Wikipedia
 - Memory Alpha
 - LinkedIn
 - None of the above
-

C.1.4. Students Introduction

Thank you very much for your interest in our study. We are researchers from XXX and are researching what kind of questions can be used to identify software developers. By taking part in our study you will help us to evaluate the difficulty of our questions for people with and without software development experience. The study should take roughly 30 minutes. There will be questions with a time constraint, so please start the survey only if you have the time to complete it in one go. Please do not share any questions or results with your fellow students. We are not testing you. We are only interested in how well the questions perform, so simply answer to the best of your ability.

Notification No Limit

The next question will measure your time. Please try to answer as quickly as you can.

Notification Countdown

You will have XX seconds to answer the next question.

C.1.5. Attack Scenario Introduction

Thank you very much for your interest in our study. We are researchers from XXX and are researching what kind of questions can be used to identifying software developers. We would like to test how difficult our questions are for people who do not program. For this we will show you 10 different questions. Please try to answer the questions correctly within the given time limit as if you were a developer. You can use any resources you like to aid you in finding the right answers. The study should take roughly 20 minutes. There will be questions with a time constraint, so please start the survey only if you have the time to complete it in one go. You will be shown an **orange** notification before each bonus question, informing you about the time limit you have to solve the task correctly. All bonus questions will show a timer with the remaining time to solve the task. Answering correctly within the time limit will earn you 2€ per question, for a maximum bonus of 20€. Thank you very much for helping us in our research.

Notification

The next question will earn you 2€ if you answer correctly within XX seconds.

Table C.1.: Time spent on the tasks and correctness in percent

Group	Task	Minimum	Median	Mean	Maximum	Correctness	Total
Base No Limit Countdown	Sources.Usage	3.89s 4.38s 6.08s	8.84s 6.93s 9.64s	10.79s 7.56s 10.53s	28.14s 12.17s 22.26s	100% 100% 100%	100%
Base No Limit Countdown	Recursive	5.42s 5s 6.32s	11.2s 11.36s 11.24s	13.13s 12.52s 13.28s	26.93s 40.78s 30.01s	100% 100% 100%	100%
Base No Limit Countdown	Boolean	3.81s 3.64s 4.3s	8.24s 7.8s 9.77s	9.42s 7.83s 11.76s	29.12s 14.33s 25.45s	100% 100% 100%	100%
Base No Limit Countdown	Rec.Lang	5.97s 4.38s 5.02s	10.2s 10.85s 10.16s	12.34s 10.49s 11.64s	29.75s 21.57s 21.05s	100% 100% 100%	100%
Base No Limit Countdown	Compiler	6.98s 5.08s 6s	12.46s 9.68s 12.94s	23.51s 10.62s 16.5s	112.40s 19.73s 40.01s	100% 96.15% 100%	98.65%
Base No Limit Countdown	IDE.Known	5.33s 5.74s 5.89s	15.99 10.19s 14.57s	23.3 11.85s 18.65s	102.95s 24.06s 60.21s	95.45% 100% 96%	97.3%
Base No Limit Countdown	RecursiveFunction3	4.16s 3.39s 4.81s	8.63s 7.5s 11.4s	16.03s 10.06 15.51s	166.24s 58.98 42.56s	100% 100% 92%	97.3%
Base No Limit Countdown	RecursiveFunction4	5.11s 4.45s 5.42s	13.2s 9.37s 19.19s	21.96s 10.82s 23.93s	116.93s 29.39s 64.13s	100% 96.15% 96%	97.3%
Base No Limit Countdown	Array	8.46s 6.52s 8.67s	16.68s 12.86s 17.26s	21.39s 14.49s 21.35s	71.12s 39.67s 54.6s	95.65% 92.31% 96%	94.59%
Base No Limit Countdown	Prime	7.13s 4.51s 16.7s	44.07s 35.33s 49.73s	84.23s 51.6s 62.93s	516.97s 232.98s 165.35s	100% 88.46% 96%	94.59%
Base No Limit Countdown	RecursiveFunction1	4.28s 5.53s 6.91s	17.15s 11.9s 18s	29.38s 13.81s 24.38s	291.57s 36.09s 90.02s	95.65% 96.15% 92%	94.59%
Base No Limit Countdown	Sorting.Array	16.51s 14.04s 17.55s	52.71s 31.55s 53.16s	72.32s 37.83s 70.51s	347.15s 113.05s 171.65s	91.3% 96.15% 92%	93.24%
Base No Limit Countdown	RecursiveFunction2	4.37s 3.08s 7.92s	12.37s 10.63s 18.81s	21.93s 32.22s 20.32s	145.76s 521.47s 62.31s	86.96% 88.46% 100%	91.89%
Base No Limit Countdown	Return	15.18s 11.71s 13.39s	35.03s 25.34s 34.89s	40.82s 28.14s 37.41s	87.81s 66.41s 80.02s	91.30% 92.31% 92%	91.89%
Base No Limit Countdown	IDE.Used	6.33s 4.89s 7.1s	9.87s 10.8s 14.82s	17.08s 11.89s 15.81s	60.58s 28.73s 30.03s	81.82% 92.31% 96%	90.41%
Base No Limit Countdown	Error.Syntax	3.87s 10.34s 13.97s	31.32s 23.53s 27.74s	52.72s 24.16s 32.64s	511.67s 43.09s 60.02s	91.3% 88.46% 88%	89.19%
Base No Limit Countdown	Palindrome	11.2s 8.95s 14.39s	51.26s 41.47s 72.23s	74.94s 51.59s 70.79s	262.31s 144.06s 116.01	82.61% 92.31 88%	87.84%
Base No Limit Countdown	CountString	4.32s 4.7s 17.81s	37.81s 34.24s 44.99s	49.24s 48.88s 48.75s	181.63s 366.64s 104.78s	86.36% 80.77% 92%	86.30%
Base No Limit Countdown	Backward.Loop	21.56s 18.22s 28.08s	79.10s 48.79s 60.56s	273.56s 59.19s 74.08s	4411.88s 344s 219.05s	82.61% 80.77% 88%	85.14%
Base No Limit Countdown	Error.OutOfBounds	13.21s 5.34s 18.66s	39.77s 28.18s 34.72s	63.62s 32.11s 47.51s	293.29s 67.18s 91.20s	91.3% 81.48% 76%	82.43%
Base No Limit Countdown	Infinite.Loop	7.47s 5.56s 24.89s	50.92s 53.99s 65.91s	97.48s 67.85s 86.7s	542.42s 236.57s 180.12s	78.26% 80.77% 84%	81.08%
Base No Limit Countdown	Parameter	3.96s 4.13s 17.52s	41.27s 28.35s 44.67s	95.27s 26.68s 43.98s	890.67s 71.9s 60.12s	86.96% 80.77% 68%	78.38%

D. Appendix: Deception: Study Announcement in a Password-Storage Study

D.1. Security Requests

Figure D.1 visualizes the security request procedure. When we were sent a solution where the password was stored in plain text or where the participant received less than 6 points in the *security score*, we sent the following messages:

- **SecRequest-P:** Participant handed in plain text code:
R: *I saw that the password is stored in clear text. Could you also store it securely?*
- **SecRequest-G:** Security score < 6:
R: *Thank you for submitting your solution. Now I have one further request. I noticed, that you did not follow industry best practices, e.g., NIST (National Institute of Standards and Technology) or OWASP (Open Web Application Security Project), to securely store the end-user password. Could you please revise your submission and ensure that you follow industry best practices? You can find some information on OWASP on this website: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md and information on NIST on this website: <https://pages.nist.gov/800-63-3/sp800-63b.html> in section 5.1.1.2.*

Participant	Prompting	Framework	Payment	Include SecRequest-G	Function	Length in bits	Iteration	Salt	Secure	Score	Copied	NIST	OWASP
FJN1	0	JSF	120	1 Day	SHA-1	160	1		1	2			✓
FJN3	0	JSF	120	1 Day	bcrypt	184	2 ¹⁰	SR	1	6			
FJN5	0	JSF	220	50 min	PBKDF2 (SHA-1)	256	10 000	St	1	5	✓	✓	
FJN7	0	JSF	120	1 Day	MD5	128	1	SR	1	4			
FJN8	0	JSF	120	2 Days	PBKDF2 (SHA-512)	512	65 536	St	1	5			✓
FJN9	0	JSF	220	19 Days	Argon2i	256	2	SR	1	7	✓	✓	
FJN10	0	JSF	220	1h 30min	PBKDF2 (SHA-1)	128	65 536	SR	1	5			✓
FJN12	0	JSF	220	15min	Argon2i	256	10	SR	1	7			✓
FJP2	1	JSF	120	2 Days	PBKDF2 (SHA-1)	512	1 000	SR	1	5.5			
FJP3	1	JSF	120	5h 30min	Argon2i	256	20	SR	1	7			✓
FJP6	1	JSF	120	2 Days	Argon2i	256	40	SR	1	7	✓		
FJP7	1	JSF	120	35min	PBKDF2 (SHA-1)	128	4	SR	1	4	✓	✓	✓
FJP8	1	JSF	120	3h 30min	PBKDF2 (SHA-1)	512	1 000	SR	1	5.5	✓	✓	
FJP9	1	JSF	120	2 Days	PBKDF2 (SHA-1)	512	1 000	SR	1	5.5	✓	✓	
FJP10	1	JSF	120	1 Day	Argon2i	256	40	SR	1	7	✓		
FJP11	1	JSF	220	1 Day	bcrypt	184	2 ¹²	SR	1	6			
FSN1	0	Spring	120	2 Days	Argon2i	256	20	SR	1	7	✓	✓	✓
FSN3	0	Spring	120	6 Days	Argon2i	256	4	SR	1	7	✓		✓
FSN5	0	Spring	120	4 Days	bcrypt	184	2 ¹⁰	SR	1	6			✓
FSN7	0	Spring	120	1 Day	Argon2i	256	40	SR	1	7	✓		
FSN9	0	Spring	220	1 Day	bcrypt	184	2 ¹⁰	SR	1	6		✓	✓
FSN10	0	Spring	120	1h	Argon2i	256	40	SR	1	7	✓	✓	
FSP2	1	Spring	120	10h	PBKDF2 (SHA-1)	128	65 536	SR	1	5			
FSP4	1	Spring	120	3 Days	bcrypt	184	2 ¹⁰	SR	1	6			
FSP6	1	Spring	120	14h	Argon2d	128	3	SR	1	7	✓		

Table D.1.: Evaluation of participants' submissions after SecRequest-G

Include SecRequest-G: Time participants needed to add security after SecRequest-G. **Salt:** SR = SecureRandom, R = Random, St = Static. **Copied:** Security code was probably copied and pasted from the Internet. **NIST/OWASP:** Participant stated that he/she followed the security guidelines of NIST/OWASP programming the task.

D.2. Security Scale

We based the evaluation of participants' submissions on the security score of Naiakshina et al. [130]:

1. The end-user password is salted (+1) and hashed (+1).
2. The derived length of the hash is at least 160 bits long (+1).
3. The iteration count for key stretching is at least 1 000 (+0.5) or 10 000 (+1) for PBKDF2 and at least $2^{10} = 1 024$ for bcrypt (+1).
4. A memory-hard hashing function is used (+1).
5. The salt value is generated randomly (+1).
6. The salt is at least 32 bits in length (+1).

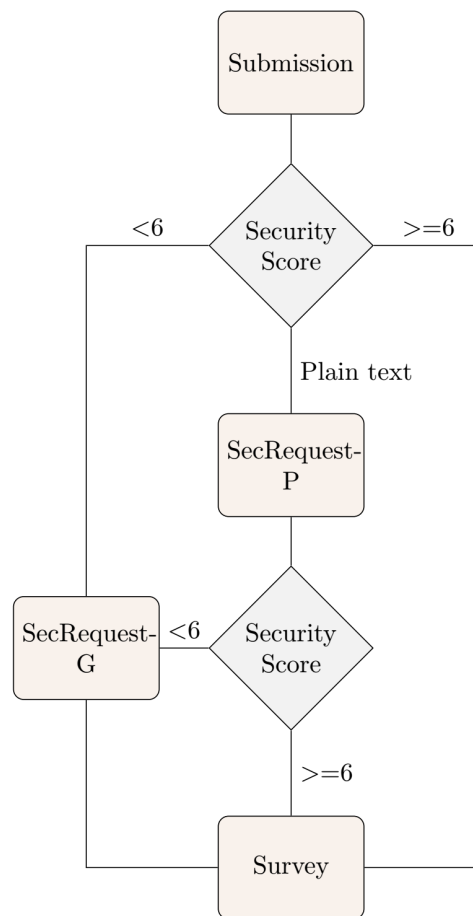


Figure D.1.: Security request procedure

Participants received SecRequest-P, if they submitted a plain text solution and SecRequest-G if the solution scored less than 6 points. Participants with plain text solutions thus, could receive both requests.

H	Sub-sample	IV	DV	Test	O.R.	CI	p-value	cor - p-value
H-P1 [#]	-	Prompting	Secure	FET	6.51	[1.51, 33.18]	0.006 [*]	0.01 [*]
H-G1 [#]	-	Java experience	Score	Kruskal-Wallis	-	-	0.75	0.75
H-G2 [#]	-	Stored passwords before	Secure	FET	0.42	[0.01, 8.63]	0.59	0.59
H-F1 [#]	secure = 1	Framework	Score	Wilcoxon rank sum	-	-	0.16	0.32
E-A1	secure = 1	Java experience	Score	Kruskal-Wallis	-	-	0.80	-
E-A2	-	Framework	API usability	Wilcoxon rank sum	-	-	0.08	-
E-A3	-	Score	API usability	Pearson Cor.	-	[-0.21, 0.38]	0.57	-
S-C1	secure = 1 & group = JSF	Study sample	Score	Wilcoxon rank sum	-	-	0.73	-
S-C2	group = JSF	Study sample	Implementation time initial submission	Wilcoxon rank sum	-	-	0.83	-
S-C3	group = JSF	Study sample	Implementation time time for SecRequest-P	Wilcoxon rank sum	-	-	0.52	-

Table D.2.: Summary of statistical analysis

IV: Independent variable, DV: Dependent variable, O.R.: Odds ratio, CI: Confidence interval, E-A: Exploratory analysis, S-C: Sample Comparison. All tests were conducted on security values of the *initial solutions* before participants received any security requests. H-P1 and H-G2 as well as H-G1 and H-F1 are corrected with Bonferroni-Holm correction (cor-p value). # = Hypothesis of Naiakshina et al. [131, 132], * = Significant Tests.

D.3. Hypotheses

Due to the adjusted study design, we were able to test only four of the seven main hypotheses from [131]. While it was possible to track security attempts in a lab setting, in an online study this information was not accessible. We did, though, consider the subset *secure = 1* (achieving security) for our analysis. Hypotheses from [131]:

- H-P1 - Priming has an effect on the likelihood of participants attempting security.
- H-F1 - Framework has an effect on the security score of participants attempting security.
- H-G1 - Years of Java experience have an effect on the security scores.
- H-G2 - If participants state that they have previously stored passwords, it affects the likelihood that they store them securely.

D.4. Summary of Statistical Analysis

Table D.2 summarizes all hypotheses from our analysis of the study.

D.5. Playbook

We used the same playbook Naiakshina et al. used in [132]. We extended and adapted it by several relevant aspects. **P** indicates the participant and **R** indicates the researcher. Because of space limitation, we mention only playbook extensions here.

D.5.1. Study Announcement and Study Offer

We are researchers from the University of Bonn working in the field of software usability. We are always looking for software developers and system administrators who are interested in taking part in one of our studies (programming and surveys). All data will be processed pseudonymously and stored anonymized after the study; there will be no identifying information published in any form. If you are interested in participating in studies - Please contact us!

After the participants placed a bid on the project, we contacted them via the private chat at Freelancer.com with the following message:

R: Hello XYZ, we are happy that you want to take part in our study. The payment will be divided into 3 milestones: 50 euros (100 euros) for your initial code release, additional 50 euros (100 euros) for the final code release after our review and further additional 20 euros for completing our survey about your programming experience and your experiences with this task. If this is fine for you and you want to take part in the study, we need you to sign a consent form. Please go to the following website to do so: LINK Your study-ID is: XXX Thanks in advance. Kind regards, ...

When the freelancer signed the consent form, we sent the second message and a ZIP file with task and code:

R: Hello XYZ, you agreed to take part in our study. Thank you for signing the consent form! Now I will send you the code as a ZIP file. You will find the task in there too. Please have a look at it and tell me if you want to do it. Then I will award you with the project and create the milestones. Kind regards, ...

After that message, we got mostly three kinds of reactions:

- The freelancer agreed to take part, we awarded him/her and wished him/her *Happy coding!*
- The freelancer did not react anymore:
Hello XYZ, what do you think? Are you still interested to take part in the study?
- **P:** *May I check the code and get back to you tomorrow/after/...?*
R: *Yes, sure! Take your time.*

D.5.2. Deadline

Some of the participants asked us for a final deadline. Since we did not want to rush them, we did not set one, but asked them to tell us how much time they needed to finish it. When a freelancer did not ask for a deadline, we decided to contact him/her after 10 days to ask for an update.

We had two cases where the freelancers did not answer several questions for updates. In that case we set a deadline and ended the study, when they exceeded it.

- **P:** *What is the final delivery?*
R: *What do you think how much time you need to solve the task?*
P: DATE
R: *Ok, that's fine. Thank you.*
- Deadline exceeded:
R: *Hey XYZ, could you give us a status update? Kind regards, ...*
- 10 days after task was sent and in case no deadline was set:
R: *Hey XYZ, could you give us a status update? What do you think how much time you need to solve the task?*
- If no reaction after 3 weeks:
R: *Hello, please send your solution till date in one week. If you decided to no longer participate in the study, I would be very glad if you could let me know. Thank you and kind regards, ...*

D.5.3. Password-related Questions

The following questions concerned password storage:

- Before submitting initial solution:
P: *Should I implement security/secure password storage?*
R: *Whatever you would recommend!*
- **P:** *Is *** fine?*
R: *Whatever you would recommend/use!*
- **P:** *I cannot find password encryption in the requirements, can you tell me where it is written? I might have missed it.*
R: *It is not in there, that is true. But could you add it?*
- After SecRequest-G:
P: *Should I implement all the rules mentioned in NIST document? There are so many rules in NIST.*
R: *You don't have to implement all rules, but please concentrate on secure password storage in a database on the back-end site. For us it's most important that the password is saved securely.*

D.5.4. General Questions

Also in the general communication we often received similar questions.

- **P:** *Can I build it from scratch?*
R: *You can solve the task as you prefer.*
- **P:** *If I have some questions for your project, can I ask you?*
R: *Sure!*
- **P:** *Do you have a server where we can upload this code for you to test?*
R: *None that I have access to. Would it be possible for you to send me a video or screenshots so I can see that it is working on your computer?*
- Participant did not work on our database:
R: *Could you also make it work on our database?*
- **P:** *Once the user registers, do we need to send verification email also and once he clicks on that, we will make user status as active?*
R: *No, we only need the data to be stored in our data base for now!*
- **P:** *I need to see the ER diagram.*
R: *We do not have that yet. Is it necessary?*
P: ...
R: *Could you please create a single table for now and I will talk to my mentor about the rest?*
- **P:** *Do you require the login functionality as well? Should I implement is as a further task? What else except registration will be needed?*
R: *No, thank you. Please only program the registration functionalities.*
- **P:** *The password is in the database, so users won't be able to access it.*
R: *And what if someone gets access to the database?*
- **P:** *Could you tell me what (...) is for? / Could you help me with (...)?*
R: *Since we are conducting a study and all participants should have the same requirements, I cannot help you with specific questions about the code. I'm sorry!*

- Participant is confused (after SecRequest-G):

R: *You will not receive any further request. You can choose, which industry standard you would like to follow; OWASP or NIST. So that you do not have to read the whole NIST guideline, you can read all necessary information in section 5.1.1.2. This section approximately complies with the length of the provided OWASP source. It is up to you to choose one standard. Afterwards you only have to fill out a subsequent survey, which concludes the study.*

D.5.5. Receiving the Solution and Survey Request

After receiving the final solution, we wrote:

Thank you, for sending your result! I will look into it.

We checked the remote database for code examples. If the freelancer had not worked on it, we wrote: *Could you also make it work on our database?*

If the freelancer could not make it work on our database, we asked for pictures and a video that showed that the code was working. We also checked it for functionality and if everything worked, we asked the freelancers to take part in our survey.

Message: *Hello XYZ, thank you for sending us your results. Like announced in the study description we would like to invite you to a concluding survey. You can find it here: [LINK](#) Kind regards, ...*

D.5.6. Exit Communication

After the freelancers finished the survey, we released the last milestone and sent them the following message: *Thank you for your participation! We are happy about your feedback, but we would like to kindly ask you to not mention our study content in order to ensure the validity of our study. Thank you again! And they all replied that they would not mention it.*

Many of the freelancers asked for a good or a five-star review, which we gave them: *Yes, we did. It was nice working with you.*

Also many of them asked, if we had further projects in which they could take part.

R: *At the moment we unfortunately have only one project.*

D.5.7. Review

We gave all participants the same review:

Very good communication, delivered on time. It was nice working with him/her!

E. Appendix: Task Design: Code-Reviewing and Code-Writing

E.1. Survey

1. Thank you very much for your interest in our study. We are researchers from the University of Bonn. We are investigating the process of code reviewing for software developers. By taking part in our study you will help us understand how developers explore issues in source code written by other developers.

Since we will be showing you code snippets please make sure to participate in the survey using your computer instead of a mobile device.

2. You and your colleagues are developing a social networking website to share pictures with family and friends. People need to register to this website in order to share their pictures. The front end has already been developed using the Model-view-controller (MVC) architectural pattern. A colleague of yours committed the following class and you have been asked to do a code review for it, before it gets accepted and released.

The class implements a registration functionality in the web application: a user with the following attributes (email, name, gender, birth date, username and password) is stored in the database.

Please review the snippet as you would for a company. You get to decide if this code is deployed or if it goes back to the developer for improvement. In both cases please explain your decision. You can review the code manually or use tools if you prefer. (Prompting: Please ensure that the user password is stored securely.)

[CODE SNIPPET]

3. What did you check for? *[Text box]*
4. Please write your review for your colleague. Please suggest actions for all issues. *[Text box]*
5. Do you approve the release of the code?
 - The code has passed the review. The code can be released
 - The code can be released but the issues mentioned above need to be fixed for the next update.
 - Code did not pass review, please fix issues as mentioned above.
6. Did you check for security? *[Yes; No]*
7. Did you ensure that the user password was stored securely? *[Yes; No]*
8. Can you please explain what hashing and salting for passwords is? *[Text box]*
9. How did you review the code snippet? *[Manually; Using the following tools: [Text box]*
10. In the past I have reviewed code written by others. *[Yes; No]*
11. If Yes: How many times have you reviewed code written by others in the past year? *[Text box]*
12. What percentage of your code reviewing time do you dedicate to security? *[Text box]*
13. What percentage of your programming time do you dedicate to security? *[Text box]*
14. I have a good understanding of security concepts. *1 - Strongly Disagree - 7 Strongly Agree*
15. Please rate the following items: *1- Never - 7 Always*
 - How often do you ask for help when faced with security problems?

- How often are you asked for help when others are faced with security problems?
16. I feel responsible for the security of end-users when writing code. *1 - Strongly Disagree - 7 Strongly Agree*
 17. I feel responsible for the security of end-users when reviewing code. *1 - Strongly Disagree - 7 Strongly Agree*
 18. Please enter your age: *[Text box]*
 19. Please select your gender. *[Male; Female; Prefer not say; Prefer to self-describe: Text box]*
 20. What is your current occupation? *[Freelance developer; Industry developer; Freelance tester; Industry tester; Academic researcher; Undergraduate student; Graduate student; Other:]*
 21. What type(s) of software do you develop/test? (Multiple answers possible) *[Web applications; Mobile/App applications; Desktop applications; Embedded Software Engineering; Enterprise applications; Other (please specify):]*
 22. In which country do you mainly work / study? *[Text box]*
 23. How many years of experience do you have with software development in general?
 24. How many years of experience do you have with Java development?
 25. How many people work in your team? Please enter 1 if you work on your own.
 26. Please select what is more important to you. *[Functionality - Security (Slider between both, Middle: Equally important)*

```

1   main{
2       print(func("hello world"))
3   }

5   String func(String in){
6       int x = len(in)
7       String out = ""
8       for(int i = x-1; i >= 0; i--){
9           out.append(in[i])
10      }
11      return out
12  }
13

```

Figure E.1.: Test for software developing skills [59]

27. Please select the returned value of the pseudo code above [see Figure E.1]:
 - hello world hello world hello world hello world
 - world hello
 - hello world
 - hello world 10
 - HELLO WORLD
 - dlrow olleh
 28. As a non-profit academic institution we are interested in offering fair compensation for your participation in our research. How do you rate the payment of the study? *[Way too little; Too little; Just right; Too much; Way too much;]*
-

29. How many minutes did you actively work on this survey?
30. Thank you for taking part in our study! We really appreciate your time and effort. We hope our results will help improving security awareness in code reviewing. If you have any comments or suggestions, please leave them here and then please click on "Continue" to complete the survey.

E.2. Playbook

During the study we conducted a playbook to ensure all participants received the same information. When a seller contacted us, there were three cases: the offer is the correct amount, the offer is too expensive or the offer is too cheap.

- Hello! Thank you for your interest. We would be delighted to have you participate in our java code reviewing study. Do you have experience programming in Java? If you agree to proceed we would send you a link, from which you can then complete our online survey. We expect the survey to take no more than two hours. To complete the survey you would have a week. Would you like to proceed? Kind regards, XXX
- If the offer was not \$50 we added the following question:
- If you would like to participate could you increase your payment requirement and send us a custom offer of \$50?
or
We do however have a budget of \$50 per participant. If you would like to participate could you send us a custom offer of \$50?
- Once the participants had sent us a custom offer, they received the answer:
Thank you! I will confirm your offer and then send you a link to the survey
- When you have completed the survey, we would appreciate it if you do not write any specific comments regarding the survey in your rating of us on Fiverr. As the study is currently ongoing this can lead to inconsistent results. Thank you for your understanding!

Below is a list of questions we were asked and our responses to them (P = Participant):

- P: I have very little programming experience in Java albeit.
Us: We are looking for people with experience programming in Java. If you feel you fulfill this requirement you are welcome to take part.
 - P: Is clicking on that link mean that I must start?
Us: You should be able to continue where you left off, if you happen to want to continue the survey at a later point.
 - P: I hope that the answers are to be in English?
Us: Yes the survey is in English.
 - P: I don't even know what is the problem and what is it about your research?
Us: This is a Java code reviewing study. You will be required to complete a code review and then answer some questions.
 - P: Why is that obligatory? (to get paid)
Us: It is important for the study that each participant is treated the same.
-

-
- P: How many files / classes are and LOC (Lines of Codes) will be there in the code base? (roughly)
Us: There are three files, two with roughly 100 lines of code and the third with 15.
 - P: Do these two hours have to be without intervals?
Us: You're welcome to take breaks as and when you need them.
 - P: Just to get to know, do we need to do the survey straightaway for 2 hours or can we save the part that we have done and continue it later?
Us: You don't need to complete the survey immediately. You can complete it at any point in the week after your offer is accepted.
 - P: No personal information?
Us: All data will be processed pseudonymously and stored anonymized after the study; there will be no identifying information published in any form.
 - P: Seems a little sketchy to be honest, I'd like to make sure this is legit.
Us: If you would like to participate could you send us a custom offer of \$50? We would then accept your offer and send you the link, thus ensuring no risk for you.
 - P: I wish to complete your online survey but Unfortunately paying you \$50 is stopping me to participate.
Us: You would be receiving the money.
 - P: No I don't have any experience in Java.
Us: Ok, thank you for your response!
 - P: But how you know I take a survey and how you pay me?
Us: You have sent us an offer of \$50. I would confirm this offer and send you a link to our survey. Upon completion you will get paid.
 - P: But I don't have any project if I don't deliver how it is possible to send money?
Us: You have sent us an offer. As mentioned, I would confirm this offer, send you the link to our survey, which you would then complete. You would then confirm that you have delivered the service. We would then check that you have completed the survey. If this is the case, we will confirm completion and you will receive your payment.
 - P: And what is the deadline? Is it limited by time?
Us: You have a week to complete the survey.
 - Participant claims to be finished, but the response is not submitted.
Us: We have not received your response. Can you check that you have completed the survey?
 - Participant mentioned word 'security' in review.
Us: Thank you very much for your kind review. We have however noticed that you mentioned the word "security" in your review. As this study is ongoing, we would rather not have any comments regarding security on our profile. Is it possible you could change your review message? Kind regards, XXX
 - P: They are asking for my review will you give me review otherwise I will mention that you haven't given me review after all work.
Us: I have completed your review already.
 - P: Please tell me and type here what review you want from me as seller.
Us: Telling you what to review us is not in compliance with Fiverr's terms and conditions. We
-

would appreciate it if you do not mention any specifics to the survey, but you are welcome to comment on the experience as a whole working with us.

- P: Do you have something new for me?
Us: I'm afraid we don't have any more work for you at this time.
- P: What was the survey for? (After completion)
Us: We are researchers working in the field of software usability. The survey is to be used to better understand how freelancers work and what benefits and disadvantages a code review has.
- By reapplication: Thank you for your interest in our survey, unfortunately we need new participants for the survey.
- Review: Very good communication, delivered on time. It was nice working with *name*!

E.3. Code Snippets

Participants were shown at random one of three insecure code snippets. The code snippets for the study can be found here:

Plaintext

<https://gist.github.com/u-cec/54e79635ec44234f8aa8ae4514d3d9e9>

MD5

<https://gist.github.com/u-cec/d25963ac45569962fca2291661f6e2f8>

Base64

<https://gist.github.com/u-cec/3fedf84f64918d9cafab61042cee8658>

E.4. Evaluation of Participants' Code Reviews

Table E.1 shows an overview of the evaluation of participants' submissions.

E.5. Participants' Review Criteria

Criteria mentioned by participants are summarized in Table E.2.

E.6. Found Password Storage Issue

Participants who reported issues with the code are summarized in Table E.3.

Participant	Code Snippet	Prompted	Survey duration [minutes]*	Time for review [minutes]	Found password storage issue	Ready for release?	Review word count
NB1	Base64	n	31	18	0	✗	127
NB2	Base64	n	86	26	0	✗	56
NB3	Base64	n	24	4	0	✗	17
NB4	Base64	n	316	44	0	✓(!)	87
NB5	Base64	n	34	14	0	✗	128
NB6	Base64	n	29	12	0	✗	205
NB7	Base64	n	23	11	0	✗	63
NB8	Base64	n	36	27	0	✗	261
<hr/>							
NM1	MD5	n	6119	30	0	✗	41
NM2	MD5	n	12	3	0	✓(!)	33
NM3	MD5	n	326	3	0	✗	443
NM4	MD5	n	62	38	0	✗	177
NM5	MD5	n	83	50	0	✓	117
NM6	MD5	n	254	195	0	✓(!)	79
NM7	MD5	n	85	49	0	✓(!)	40
<hr/>							
NP1	Plaintext	n	83	37	1	✓(!)	74
NP2	Plaintext	n	151	118	0	✗	228
NP3	Plaintext	n	37	19	0	✗	73
NP4	Plaintext	n	3511	582	0	✗	171
NP5	Plaintext	n	40	29	1	✗	157
NP6	Plaintext	n	13	3	0	✓(!)	24
<hr/>							
PB1	Base64	p	206	102	1	✗	265
PB2	Base64	p	934	46	1	✗	232
PB3	Base64	p	87	50	0	✓(!)	28
PB4	Base64	p	12	2	0	✓(!)	6
PB5	Base64	p	2407	132	0	✗	77
PB6	Base64	p	9045	38	0	✓(!)	82
PB7	Base64	p	68	41	0	✓(!)	30
PB8	Base64	p	198	122	0	✓(!)	70
<hr/>							
PM1	MD5	p	25	14	1	✗	82
PM2	MD5	p	5798	1	0	✓	45
PM3	MD5	p	35	14	0	✗	45
PM4	MD5	p	129	105	1	✓(!)	148
PM5	MD5	p	6153	55	1	✓(!)	47
PM6	MD5	p	66	15	1	✗	84
PM7	MD5	p	39	15	1	✗	49
<hr/>							
PP1	Plaintext	p	23	3	1	✗	25
PP2	Plaintext	p	2490	2	1	✓(!)	42
PP3	Plaintext	p	4444	67	1	✓(!)	87
PP4	Plaintext	p	25	12	1	✗	85
PP5	Plaintext	p	76	38	0	✗	107
PP6	Plaintext	p	198	14	0	✓(!)	89
PP7	Plaintext	p	53	5	0	✗	36
PP8	Plaintext	p	5910	21	0	✓(!)	68

Table E.1.: Evaluation of participants' code reviews

* Some participants started the survey and probably left it for some days since the deadline was to complete it within one week. ✗: Code did not pass review, please fix issues as mentioned above. ✓(!): The code can be released but the issues mentioned above need to be fixed for the next update.

✓: The code has passed the review. The code can be released. **Found password storage issue:** insecure password storage was mentioned as an issue in the review.

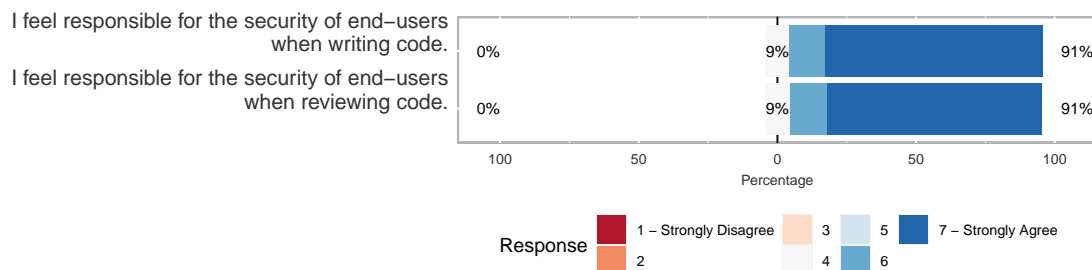


Figure E.2.: The responses on whether the participants feel responsible for security while code reviewing and writing (group: prompted)

	Criteria	Total Count	Participants	
			Prompted	Not prompted
Implementation	Functionality	6	PM4, PB6, NM5	NB6, NB1, NP1
	Logic	8	PM2, PM3, PP5, PB7	NB4, NP3, NB6, NM6
	Maintainability	1		NB2
	Performance / Efficiency	2	PB5	NP4
	Error Handling	14	PB1, PP3, PB3, PM3, PM5, PP5, PM7	NB1, NM2, NM4, NM5, NB6, NP4, NB8
Testing and Bugs	Quality Assurance	1		NM3
	Unit tests	1	PB7	
	Bugs/ Errors in the code	3		NM1, NM5, NP4
	Syntax	10	PM1, PB3, PM4, PP5, PB7, PP8, PM7	NM6, NP1, NP3
Standards and Validation	Code standards	2	PB1	NB1
	Code format	3	PB2, PP5	NM4
	Correct usage of get and set methods	9	PM1, PB3, PM5, PP5	NM2, NP1, NB3, NB4, NB5
	Model view controller architecture	3		NM1, NP2, NB8
	Imported Packages and Libraries	5	PP3, PB6	NP2, NB1, NB8
	Input validation	6	PM6, PP7, PP5	NB6, NM7, NM2,
	Null checks	1	PM6	
	Style issues	7	PB2, PM4, PP5, PP8	NB1, NP3, NP4, NB7
	Duplicated/ Unused code	2	PB1, PB2	
	Code complexity	4	PM4, PP5, PB5, PB7	
	Readability	5	PM4, PB5	NM1, NM4, NM5
Security	Comments	5	PB1, PM4, PB7	NB4, NP4
	Security	10	PB1, PP2, PP8, PM7, PM4, PP4	NM4, NP3, NP4, NP6
	Password Storage Security	11	PP1, PM1, PP2, PB2, PB6, PM6, PB7, PB8, PM7	NP1, NM4,
	Data Security	4	PB1, PP4, PM7	NM4

Table E.2.: All criteria mentioned by participants

	Found Issue	Total Count	Participants	
			Prompted	Not prompted
Found Password Storage Issue	Secure password storage	13	PP1, PM1, PB1, PP2, PB2, PP3, PM4, PP4, PM5, PM6, PM7	NP1, NP5
	Password validation	3	PP4, PB8	NM3
	Password encryption	4	PP1, PP2, PP3	NP5
	SQL and JAR injections	1	PB6	
	Password storage to complex	1	PM3	
Security Score	Storage sufficient	3	PB8	NM4, NM5
	Function issue	5	PM1, PM4, PP4, PM5, PM6	
Distraction tasks	Logical Mistake	8	PM3, PP5, PP7	NM3, NM4, NB5, NP4, NB8
	Exception swallowing	12	PM4, PB1, PB3, PM3, PM7, PP5	NB1, NB2, NB6, NB8, NM4, NP3

Table E.3.: Issues found by participants

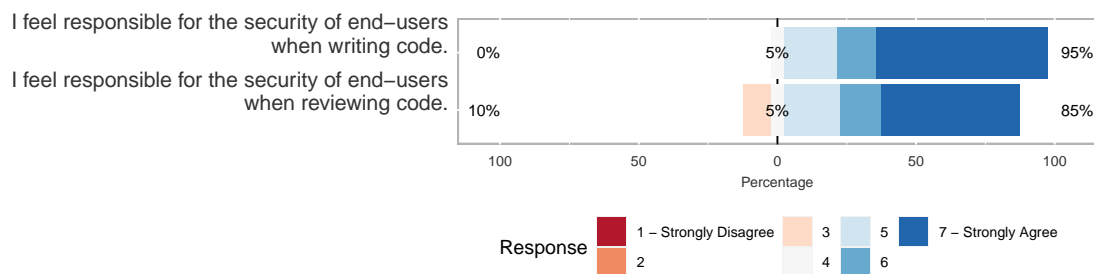


Figure E.3.: The responses on whether the participants feel responsible for security while code reviewing and writing (group: non-prompted)

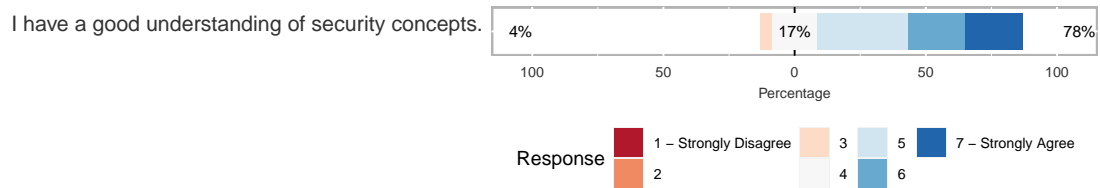


Figure E.4.: The responses on whether the participants reported to have a good understanding of security concepts (group: prompted)

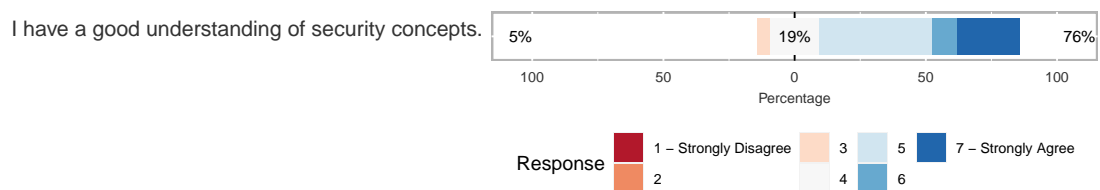


Figure E.5.: The responses on whether the participants reported to have a good understanding of security concepts (group: non-prompted)

Bibliography

- [1] Clickworker. URL <https://www.clickworker.de/>. <https://www.clickworker.de/>, Accessed: January 2020.
- [2] Ghtorrent. URL <https://ghtorrent.org/>. <https://ghtorrent.org/>, Accessed: January 2020.
- [3] Open web application security project (owasp). https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md. Last Accessed: November 14, 2019.
- [4] Fiverr.com. URL <https://www.fiverr.com/>. Accessed: September 2020.
- [5] Amazon mechanical turk (mturk). URL <https://www.mturk.com/>. Accessed: January 2020.
- [6] Notepad++. <https://notepad-plus-plus.org/>. Accessed: 28-01-19.
- [7] Qualtrics. URL <https://www.qualtrics.com>. <https://www.qualtrics.com>, Accessed: January 2020.
- [8] Xing. URL <https://www.xing.com/>. Accessed: August 2019.
- [9] Code::blocks ide. www.codeblocks.org/, 2018. Accessed: 28-01-19.
- [10] Hello, colaboratory - colaboratory - google. <https://colab.research.google.com/>, 2018. Accessed: 28-01-19.
- [11] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, Piscataway, NJ, USA, May 2016. IEEE Press. doi: 10.1109/SP.2016.25.
- [12] Y. Acar, S. Fahl, and M. L. Mazurek. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In *Cybersecurity Development (SecDev), IEEE*, pages 3–8, Piscataway, NJ, USA, 2016. IEEE, IEEE Press.
- [13] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. Comparing the usability of cryptographic apis. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 154–171, San Jose, CA, USA, 2017. IEEE. doi: 10.1109/SP.2017.52.
- [14] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl. Security developer studies with github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 81–95, Santa Clara, CA, 2017. USENIX Association. ISBN 978-1-931971-39-3. URL <https://www.usenix.org/conference/soups2017/technical-sessions/presentation/acar>.
- [15] A. Adams and M. A. Sasse. Users are not the enemy. *Communications of the ACM*, 42(12): 40–46, 1999.

-
- [16] M. A. Ahmed and J. van den Hoven. Agents of responsibility—freelance web developers in web applications development. *Information Systems Frontiers*, 12(4):415–424, Sep 2010. ISSN 1572-9419. doi: 10.1007/s10796-009-9201-0. URL <https://doi.org/10.1007/s10796-009-9201-0>.
- [17] D. Akhawe and A. P. Felt. 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)*, pages 257–272, Washington, D.C., 2013. USENIX. ISBN 978-1-931971-03-4. URL <https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/akhawe>.
- [18] H. Almuhiemedi, A. P. Felt, R. W. Reeder, and S. Consolvo. Your reputation precedes you: History, reputation, and the chrome malware warning. In *10th Symposium On Usable Privacy and Security ({SOUPS} 2014)*, pages 113–128, 2014.
- [19] A. Amran, Z. F. Zaaba, and M. K. Mahinderjit Singh. Habituation effects in computer security warning. *Information Security Journal: A Global Perspective*, 27(4):192–204, 2018.
- [20] J. R. Anderson. Skill acquisition: Compilation of weak-method problem situations. *Psychological review*, 94(2):192, 1987.
- [21] J. R. Anderson, R. Farrell, and R. Sauers. Learning to program in lisp. *Cognitive Science*, 8(2): 87–129, 1984.
- [22] J. R. Anderson, F. G. Conrad, and A. T. Corbett. Skill acquisition and the lisp tutor. *Cognitive Science*, 13(4):467–505, 1989.
- [23] S. Anell, L. Gröber, and K. Krombholz. End user and expert perceptions of threats and potential countermeasures. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 230–239, September 2020. URL <https://publications.cispa.saarland/3299/>.
- [24] H. Assal and S. Chiasson. Security in the software development lifecycle. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 281–296, Baltimore, MD, Aug. 2018. USENIX Association. ISBN 978-1-931971-45-4. URL <https://www.usenix.org/conference/soups2018/presentation/assal>.
- [25] H. Assal and S. Chiasson. ‘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*, pages 289:1–289:13, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-5970-2. doi: 10.1145/3290605.3300519. URL <http://doi.acm.org/10.1145/3290605.3300519>.
- [26] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering, ICSE ’13*, page 712–721. IEEE Press, 2013. ISBN 9781467330763.
-

-
- [27] B. P. Bailey, J. A. Konstan, and J. V. Carlis. 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, volume 2, pages 757–762. IEEE, 2000.*
- [28] R. Balebako, A. Marsh, J. Lin, J. I. Hong, and L. F. Cranor. The privacy and security behaviors of smartphone app developers. 2014. doi: 10.14722/usec.2014.23006.
- [29] S. Baltes and S. Diehl. Worse than spam: Issues in sampling software developers. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–6, Ciudad Real, Spain. ACM. doi: 10.1145/2961111.2962628.
- [30] T. Barik, J. Smith, K. Lubick, E. Holmes, J. Feng, E. Murphy-Hill, and C. Parnin. Do developers read compiler error messages? In *Proceedings of the 39th International Conference on Software Engineering, ICSE '17*, pages 575–585, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-3868-2. doi: 10.1109/ICSE.2017.59. URL <https://doi.org/10.1109/ICSE.2017.59>.
- [31] T. Barik, D. Ford, E. Murphy-Hill, and C. Parnin. 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*, pages 633–643, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5573-5. doi: 10.1145/3236024.3236040. URL <http://doi.acm.org/10.1145/3236024.3236040>.
- [32] J. Bau, F. Wang, E. Bursztein, P. Mutchler, and J. C. Mitchell. Vulnerability factors in new web applications: Audit tools, developer selection & languages. *Stanford, Tech. Rep*, 2012.
- [33] T. Baum, O. Liskin, K. Niklas, and K. Schneider. Factors influencing code review processes in industry. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016*, page 85–96. Association for Computing Machinery, 2016. ISBN 9781450342186. doi: 10.1145/2950290.2950323. URL <https://doi.org/10.1145/2950290.2950323>.
- [34] M. Beller, N. Spruit, D. Spinellis, and A. Zaidman. On the dichotomy of debugging behavior among programmers. In *Proceedings of the 40th International Conference on Software Engineering (ICSE'18)*, pages 572–583, 2018. doi: 10.1145/3180155.3180175.
- [35] G. R. Bergersen, D. I. Sjøberg, and T. Dybå. Construction and validation of an instrument for measuring programming skill. *IEEE Transactions on Software Engineering (IEEE Trans. Softw. Eng'14)*, 40(12):1163–1184, 2014. doi: 10.1109/TSE.2014.2348997.
- [36] D. G. Bonett. Replication-extension studies. *Current Directions in Psychological Science*, 21(6): 409–412, 2012.
- [37] J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, San Francisco, CA, USA, May 2012. IEEE. doi: 10.1109/SP.2012.49.
-

-
- [38] J. Bonneau and S. Preibusch. The password thicket: Technical and market failures in human authentication on the web. In *WEIS*, 2010.
- [39] C. Bravo-Lillo, L. F. Cranor, J. Downs, and S. Komanduri. Bridging the gap in computer security warnings: A mental model approach. *IEEE Security & Privacy*, 9(2):18–26, 2011.
- [40] C. Bravo-Lillo, S. Komanduri, L. F. Cranor, R. W. Reeder, M. Sleeper, J. Downs, and S. Schechter. 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, pages 6:1–6:12, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2319-2. doi: 10.1145/2501604.2501610. URL <http://doi.acm.org/10.1145/2501604.2501610>.
- [41] C. Bravo-Lillo, L. Cranor, S. Komanduri, S. Schechter, and M. Sleeper. Harder to ignore? revisiting pop-up fatigue and approaches to prevent it. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 105–111, Menlo Park, CA, July 2014. USENIX Association. ISBN 978-1-931971-13-3. URL <https://www.usenix.org/conference/soups2014/proceedings/presentation/bravo-lillo>.
- [42] A. Bryant and K. Charmaz. *The Sage handbook of grounded theory*. Sage, 2007.
- [43] R. Buck, M. Khan, M. Fagan, and E. Coman. 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):25–34, 2018.
- [44] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological Methods & Research*, 42(3):294–320, 2013.
- [45] D. D. Caputo, S. L. Pfleeger, M. A. Sasse, P. Ammann, J. Offutt, and L. Deng. Barriers to usable security? Three organizational case studies. *IEEE Security & Privacy*, 14(5):22–32, 2016.
- [46] K. Charmaz. *Constructing grounded theory: A practical guide through qualitative analysis*. sage, 2006.
- [47] K. Charmaz. Constructionism and the grounded theory method. *Handbook of constructionist research*, 1:397–412, 2008.
- [48] K. Charmaz. *Constructing grounded theory*. sage, 2014.
- [49] K. Charmaz and L. M. McMullen. *Five ways of doing qualitative analysis: Phenomenological psychology, grounded theory, discourse analysis, narrative research, and intuitive inquiry*. Guilford Press, 2011.
- [50] Y. Chen, S. Oney, and W. S. Lasecki. Towards providing on-demand expert support for software developers. In *Proceedings of the 2016 Conference on Human Factors in Computing Systems (CHI'16)*, pages 3192–3203, 2016. doi: 10.1145/2858036.2858512.
-

-
- [51] M. Christakis and C. Bird. What developers want and need from program analysis: an empirical study. In *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*, pages 332–343, Piscataway, NJ, USA, 2016. IEEE, IEEE.
- [52] Clickworker, 2021. URL <https://www.clickworker.com/>. <https://www.clickworker.com/>, Accessed: September 2020.
- [53] J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.
- [54] L. M. Collins and S. T. Lanza. *Latent class and latent transition analysis: With applications in the social, behavioral, and health sciences*, volume 718. John Wiley & Sons, 2009.
- [55] T. S. W. Contributors. Spyder website. <https://www.spyder-ide.org/>, 2018. Accessed: 28-01-19.
- [56] J. Corbin, A. Strauss, and A. L. Strauss. *Basics of qualitative research*. sage, 2014.
- [57] I. T. Coyne. Sampling in qualitative research. purposeful and theoretical sampling; merging or clear boundaries? *Journal of advanced nursing*, 26(3):623–630, 1997.
- [58] A. Danilova, A. Naiakshina, J. Deuter, and M. Smith. Replication: On the ecological validity of online security developer studies: Exploring deception in a password-storage study with freelancers. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 165–183. USENIX Association, Aug. 2020. ISBN 978-1-939133-16-8. URL <https://www.usenix.org/conference/soups2020/presentation/danilova>.
- [59] A. Danilova, A. Naiakshina, and M. Smith. One size does not fit all: A grounded theory and online survey study of developer preferences for security warning types. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE'20)*, 2020. doi: 10.1145/3377811.3380387.
- [60] A. Danilova, A. Naiakshina, S. Horstmann, and M. Smith. Do you really code? designing and evaluating screening questions for online surveys with programmers. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 537–548. IEEE, 2021. doi: 10.1109/ICSE43902.2021.00057. URL <https://doi.org/10.1109/ICSE43902.2021.00057>.
- [61] A. Danilova, A. Naiakshina, A. Rasgauski, and M. Smith. Code reviewing as methodology for online security studies with developers - a case study with freelancers on password storage. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, pages 397–416. USENIX Association, Aug. 2021. ISBN 978-1-939133-25-0. URL <https://www.usenix.org/conference/soups2021/presentation/danilova>.
- [62] A. Danilova, S. Horstmann, A. Naiakshina, and M. Smith. Testing time limits in screener questions for online surveys with programmers. In *TBP in 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. ACM, 2022.
-

-
- [63] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson. Sometimes you need to see through walls: A field study of application programming interfaces. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, CSCW '04*, pages 63–71, New York, NY, USA, 2004. ACM. ISBN 1-58113-810-5. doi: 10.1145/1031607.1031620. URL <http://doi.acm.org/10.1145/1031607.1031620>.
- [64] D. M. DeJoy, K. R. Laughery, and M. S. Wogalter. *Warnings and risk communication*. Taylor & Francis, 1999.
- [65] N. Denzin. The research act in sociology (london, croom helm). *Denzin The Research Act in Sociology 1970, 1970*.
- [66] N. K. Denzin. *The research act: A theoretical introduction to sociological methods*. Routledge, 2017.
- [67] D. A. Dillman. *Mail and Internet surveys: The tailored design method—2007 Update with new Internet, visual, and mixed-mode guide*. John Wiley & Sons, 2011.
- [68] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, et al. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*, pages 475–488, 2014.
- [69] Eclipse. Eclipse. <https://www.eclipse.org/>, 2018. Accessed: 28-01-19.
- [70] A. Edmundson, B. Holtkamp, E. Rivera, M. Finifter, A. Mettler, and D. Wagner. An empirical study on the effectiveness of security code review. In J. Jürjens, B. Livshits, and R. Scandariato, editors, *Engineering Secure Software and Systems*, pages 197–212, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-36563-8.
- [71] S. Egelman, L. F. Cranor, and J. Hong. 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*, pages 1065–1074. ACM, 2008.
- [72] S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov, and C. Herley. Does my password go up to eleven?: The impact of password meters on password selection. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2379–2388, New York, NY, USA, 2013. ACM, ACM.
- [73] S. Elo and H. Kyngäs. The qualitative content analysis process. *Journal of advanced nursing*, 62(1):107–115, 2008.
- [74] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith. 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*, pages 50–61, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1651-4. doi: 10.1145/2382196.2382205. URL <http://doi.acm.org/10.1145/2382196.2382205>.
- [75] S. Fahl, M. Harbach, Y. Acar, and M. Smith. On the ecological validity of a password study. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, page 13. ACM, 2013.
-

-
- [76] J. Feigenspan, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg. Measuring programming experience. In *2012 20th IEEE International Conference on Program Comprehension (ICPC'12)*, pages 73–82. IEEE, 2012. doi: 10.1109/ICPC.2012.6240511.
- [77] A. P. Felt, S. Egelman, and D. Wagner. 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*, pages 33–44. ACM, 2012.
- [78] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*, page 3. ACM, 2012.
- [79] A. P. Felt, A. Ainslie, R. W. Reeder, S. Consolvo, S. Thyagaraja, A. Bettes, H. Harris, and J. Grimes. Improving ssl warnings: Comprehension and adherence. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 2893–2902. ACM, 2015.
- [80] A. Field, J. Miles, and Z. Field. *Discovering statistics using R*. Sage publications, 2012.
- [81] FindBugs. Findbugs - find bugs in java programs. <http://findbugs.sourceforge.net/>, 2018. Accessed: 28-01-19.
- [82] M. Finifter and D. Wagner. Exploring the relationship between web application development tools and security. In *Proceedings of the 2Nd USENIX Conference on Web Application Development, WebApps'11*, pages 9–9, Berkeley, CA, USA, 2011. USENIX Association. URL <http://dl.acm.org/citation.cfm?id=2002168.2002177>.
- [83] J. Finkle and J. Saba. LinkedIn suffers data breach-security experts, 2012. URL <http://in.reuters.com/article/linkedin-breach-idINDEE8550EN20120606>.
- [84] K. Finstad. Response interpolation and scale sensitivity: Evidence against 5-point scales. *Journal of Usability Studies*, 5(3):104–110, 2010.
- [85] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl. Stack overflow considered harmful? the impact of copy paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP'17)*, pages 121–136, May 2017. doi: 10.1109/SP.2017.31.
- [86] P. M. Fitts and M. I. Posner. Human performance. 1967.
- [87] J. L. Fleiss, B. Levin, and M. C. Paik. *Statistical methods for rates and proportions*. John Wiley & Sons, 2013.
- [88] D. Florêncio, C. Herley, and P. C. Van Oorschot. An administrator’s guide to internet password research. In *28th Large Installation System Administration Conference (LISA14)*, pages 44–61, 2014.
- [89] P. Fusch, G. E. Fusch, and L. R. Ness. Denzin’s paradigm shift: Revisiting triangulation in qualitative research. *Journal of Social Change*, 10(1):2, 2018.
-

-
- [90] Geany. Geany. <https://www.geany.org/>, 2018. Accessed: 28-01-19.
- [91] GitHub, 2021. URL <https://github.com/>.
- [92] B. G. Glaser and A. L. Strauss. *Discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017.
- [93] G. Glaser Barney and L. Strauss Anselm. The discovery of grounded theory: strategies for qualitative research. *New York, Adline de Gruyter*, 1967.
- [94] P. L. Gorski, L. L. Iacono, D. Wermke, C. Stransky, S. Möller, Y. Acar, and S. Fahl. 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)*, pages 265–281, Baltimore, MD, 2018. USENIX Association. ISBN 978-1-931971-45-4. URL <https://www.usenix.org/conference/soups2018/presentation/gorski>.
- [95] G*Power, 2021. URL gpower.hhu.de.
- [96] P. A. Grassi, J. L. Fenton, E. Newton, R. Perlner, A. Regenscheid, W. Burr, J. Richer, N. Lefkovitz, J. Danker, Y.-Y. Choong, et al. Nist special publication 800-63b: Digital identity guidelines. *Enrollment and Identity Proofing Requirements*. <https://pages.nist.gov/800-63-3/sp800-63b.html>, 2017.
- [97] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson. Unhappy developers: Bad for themselves, bad for process, and bad for software product. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C'17)*, pages 362–364. IEEE, 2017. doi: <https://doi.org/10.1109/ICSE-C.2017.104>.
- [98] M. Green and M. Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security & Privacy*, 14(5):40–46, 2016.
- [99] A. Hanamsagar, S. S. Woo, C. Kanich, and J. Mirkovic. Leveraging semantic transformation to investigate password habits and their causes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 570, New York, NY, USA, 2018. ACM, ACM.
- [100] J. M. Haney, M. Theofanos, Y. Acar, and S. S. Prettyman. "we make it a big deal in the company": Security mindsets in organizations that develop cryptographic products. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 357–373, 2018.
- [101] S. Hudson, J. Fogarty, C. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J. Lee, and J. Yang. Predicting human interruptibility with sensors: a wizard of oz feasibility study. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 257–264, 2003.
- [102] L. L. Iacono and P. L. Gorski. I do and i understand. not yet true for security apis. so sad. In *Proc. of the 2nd European Workshop on Usable Security, ser. EuroUSEC'17*, volume 17, 2017. doi: [10.14722/eurosec.2017.23015](https://doi.org/10.14722/eurosec.2017.23015).
-

-
- [103] JetBrains. Clion: A cross-platform ide for c and c++ by jetbrains. <https://www.jetbrains.com/clion/>, 2018. Accessed: 28-01-19.
- [104] JetBrains. IntelliJ idea. <https://www.jetbrains.com/idea/>, 2018. Accessed: 28-01-19.
- [105] JetBrains. PhpStorm: The lightning-smart ide for php programming by jetbrains. <https://www.jetbrains.com/phpstorm/>, 2018. Accessed: 28-01-19.
- [106] JetBrains. Pycharm: the python ide for professional developers by jetbrains. <https://www.jetbrains.com/pycharm/>, 2018. Accessed: 28-01-19.
- [107] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Proceedings of the 2013 International Conference on Software Engineering, ICSE '13*, pages 672–681, Piscataway, NJ, USA, 2013. IEEE. ISBN 978-1-4673-3076-3. URL <http://dl.acm.org/citation.cfm?id=2486788.2486877>.
- [108] B. Johnson, R. Pandita, J. Smith, D. Ford, S. Elder, E. Murphy-Hill, S. Heckman, and C. Sadowski. 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*, pages 73–84, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4218-6. doi: 10.1145/2950290.2950304. URL <http://doi.acm.org/10.1145/2950290.2950304>.
- [109] P.-H. Kamp, P. Godefroid, M. Levin, D. Molnar, P. McKenzie, R. Stapleton-Gray, B. Woodcock, and G. Neville-Neil. LinkedIn password leak: Salt their hide. *ACM Queue*, 10(6):20, 2012.
- [110] A. J. Kimmel. *Ethical issues in behavioral research: Basic and applied perspectives*. John Wiley & Sons, 2009.
- [111] A. J. Kimmel and N. C. Smith. Deception in marketing research: Ethical, methodological, and disciplinary implications. *Psychology & Marketing*, 18(7):663–689, 2001.
- [112] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman. Of passwords and people: Measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2595–2604, New York, NY, USA, 2011. ACM, ACM.
- [113] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey. Investigating code review quality: Do people and participation matter? In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 111–120, 2015.
- [114] K. Krombholz, W. Mayer, M. Schmiedecker, and E. Weippl. "i have no idea what i'm doing" - on the usability of deploying HTTPS. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1339–1356, Vancouver, BC, 2017. USENIX Association. ISBN 978-1-931971-40-9. URL <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/krombholz>.
-

-
- [115] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler, et al. Cognicrypt: supporting developers in using cryptography. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 931–936, Piscataway, NJ, USA, 2017. IEEE Press, IEEE.
- [116] F. Y. Kung, N. Kwok, and D. J. Brown. Are attention check questions a threat to scale validity? *Applied Psychology*, 67(2):264–283, 2018. doi: 10.1111/apps.12108.
- [117] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering, ICSE '06*, pages 492–501, New York, NY, USA, 2006. ACM. ISBN 1-59593-375-1. doi: 10.1145/1134285.1134355. URL <http://doi.acm.org/10.1145/1134285.1134355>.
- [118] P. J. Lavrakas. *Encyclopedia of survey research methods*. Sage Publications, 2008.
- [119] D. A. Linzer, J. B. Lewis, et al. polca: An r package for polytomous variable latent class analysis. *Journal of statistical software (J. Stat. Softw.)* 42(10):1–29, 2011. doi: 10.18637/jss.v042.i10.
- [120] S. H. P. Ltd. Sublime text - a sophisticated text editor for code, markup and prose. <https://www.sublimetext.com/>, 2018. Accessed: 28-01-19.
- [121] P. Mayer, J. Kirchner, and M. Volkamer. A second look at password composition policies in the wild: Comparing samples from 2010 and 2016. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 13–28, Santa Clara, CA, 2017. USENIX Association. ISBN 978-1-931971-39-3. URL <https://www.usenix.org/conference/soups2017/technical-sessions/presentation/mayer>.
- [122] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur. Measuring password guessability for an entire university. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS '13*, pages 173–186, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2477-9. doi: 10.1145/2508859.2516726. URL <http://doi.acm.org/10.1145/2508859.2516726>.
- [123] A. L. McCutcheon. *Latent class analysis*. Number 64. Sage, 1987.
- [124] A. Meyer, E. T. Barr, C. Bird, and T. Zimmermann. Today was a good day: The daily life of software developers. *IEEE Transactions on Software Engineering (IEEE Trans. Softw. Eng.)* 19, 2019. doi: 10.1109/TSE.2019.2904957.
- [125] Microsoft. Visual studio ide, code editor, vsts, & app center - visual studio. <https://visualstudio.microsoft.com/>, 2018. Accessed: 28-01-19.
- [126] M. Mora, O. Gelman, A. Steenkamp, and M. S. Raisinghani. *Research methodologies, innovations and philosophies in software systems engineering and information systems*. IGI Global Hershey, PA, 2012.
- [127] D. L. Morgan. Focus groups. *Annual review of sociology*, 22(1):129–152, 1996.
-

-
- [128] E. Murphy-Hill, C. Jaspan, C. Sadowski, D. C. Shepherd, M. Phillips, C. Winter, A. K. Dolan, E. K. Smith, and M. A. Jorde. What predicts software developers' productivity? *Transactions on Software Engineering*, 2019.
- [129] S. Nadi, S. Krüger, M. Mezini, and E. Bodden. Jumping through hoops: Why do java developers struggle with cryptography apis? In *Proceedings of the 38th International Conference on Software Engineering, ICSE '16*, page 935–946, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450339001. doi: 10.1145/2884781.2884790. URL <https://doi.org/10.1145/2884781.2884790>.
- [130] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith. 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*, pages 311–328, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4946-8. doi: 10.1145/3133956.3134082. URL <http://doi.acm.org/10.1145/3133956.3134082>.
- [131] A. Naiakshina, A. Danilova, C. Tiefenau, and M. Smith. Deception task design in developer password studies: Exploring a student sample. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS'18)*, pages 297–313, Baltimore, MD, 2018. USENIX Association. ISBN 978-1-931971-45-4. URL <https://www.usenix.org/conference/soups2018/presentation/naiakshina>.
- [132] A. Naiakshina, A. Danilova, E. Gerlitz, E. von Zezschwitz, and M. Smith. “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*, pages 140:1–140:12, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-5970-2. doi: 10.1145/3290605.3300370. URL <http://doi.acm.org/10.1145/3290605.3300370>.
- [133] A. Naiakshina, A. Danilova, E. Gerlitz, and M. Smith. On conducting security developer studies with cs students: Examining a password-storage study with cs students, freelancers, and company developers. In *Proceedings of the 2020 Conference on Human Factors in Computing Systems (CHI'20)*, pages 1–13. ACM, 2020. doi: 10.1145/3313831.3376791.
- [134] Netbeans. Netbeans. <https://netbeans.org/>, 2018. Accessed: 28-01-19.
- [135] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl. 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*, pages 1065–1077, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4946-8. doi: 10.1145/3133956.3133977. URL <http://doi.acm.org/10.1145/3133956.3133977>.
- [136] D. Oliveira, M. Rosenthal, N. Morin, K.-C. Yeh, J. Cappos, and Y. Zhuang. It's the psychology stupid: how heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 296–305, 2014.
-

-
- [137] D. S. Oliveira, T. Lin, M. S. Rahman, R. Akefirad, D. Ellis, E. Perez, R. Bobhate, L. A. DeLong, J. Cappos, Y. Brun, and N. C. Ebner. Api blindspots: Why experienced developers write vulnerable code. In *Proceedings of the Fourteenth USENIX Conference on Usable Privacy and Security*, SOUPS'18, page 315–328, USA, 2018. USENIX Association. ISBN 9781931971454.
- [138] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl. To pin or not to pin—helping app developers bullet proof their tls connections. In *24th USENIX Security Symposium (USENIX Security'15)*, pages 239–254, 2015.
- [139] S. S. or an SAP affiliate company. Sap documentation - abap workbench tools, 2018. URL https://help.sap.com/saphelp_nw73EhP1/helpdata/en/ef/d94b78ebf811d295b100a0c94260a5/frameset.htm. Accessed: 28-01-19.
- [140] L. Ponzanelli, S. Scalabrino, G. Bavota, A. Mocci, R. Oliveto, M. Di Penta, and M. Lanza. Supporting software developers with a holistic recommender system. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE'17)*, pages 94–105. IEEE, 2017. doi: 10.1109/ICSE.2017.17.
- [141] R. A. Powell and H. M. Single. Focus groups. *International journal for quality in health care*, 8(5):499–504, 1996.
- [142] L. Prechelt. Plat_Forms: A Web Development Platform Comparison by an Exploratory Experiment Searching for Emergent Platform Properties. *IEEE Transactions on Software Engineering*, 37(1):95–108, 2011. doi: 10.1109/TSE.2010.22.
- [143] L. Prechelt. Plat_forms: A web development platform comparison by an exploratory experiment searching for emergent platform properties. *IEEE Transactions on Software Engineering*, 37(1):95–108, Jan 2011. ISSN 0098-5589. doi: 10.1109/TSE.2010.22.
- [144] P. O. S. Project. Pmd- an extensible cross-language static code analyzer. <https://pmd.github.io/>, 2018. Accessed: 28-01-19.
- [145] E. M. Redmiles, Z. Zhu, S. Kross, D. Kuchhal, T. Dumitras, and M. L. Mazurek. Asking for a friend: Evaluating response biases in security user studies. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1238–1255. ACM, 2018.
- [146] R. W. Reeder, A. P. Felt, S. Consolvo, N. Malkin, C. Thompson, and S. Egelman. 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, pages 512:1–512:13, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3174086. URL <http://doi.acm.org/10.1145/3173574.3174086>.
- [147] T. Robertson, S. Prabhakararao, M. Burnett, C. Cook, J. R. Ruthruff, L. Beckwith, and A. Phalgune. Impact of interruption style on end-user debugging. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 287–294, 2004.
-

-
- [148] T. Robertson, J. Lawrance, and M. Burnett. Impact of high-intensity negotiated-style interruptions on end-user debugging. *Journal of Visual Languages & Computing*, 17(2):187–202, 2006.
- [149] S. Roth, L. Gröber, M. Backes, K. Krombholz, and B. Stock. 12 angry developers ? a qualitative study on developers? struggles with csp. In *ACM CCS 2021*, November 2021. URL <https://publications.cispa.saarland/3463/>.
- [150] S. SA. Sonarqube - the leading product for continuous code quality. <https://www.sonarqube.org/>, 2018. Accessed: 28-01-19.
- [151] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli. Modern code review: A case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP '18*, page 181–190. Association for Computing Machinery, 2018. ISBN 9781450356596. doi: 10.1145/3183519.3183525. URL <https://doi.org/10.1145/3183519.3183525>.
- [152] M. A. Schmuckler. What is ecological validity? a dimensional analysis. *Infancy*, 2(4):419–436, 2001.
- [153] C. Seale. Quality in qualitative research. *Qualitative inquiry*, 5(4):465–478, 1999.
- [154] M. Sedova. Comparing educational approaches to secure programming: Tool vs.ta. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, 2017.
- [155] S. M. Segreti, W. Melicher, S. Komanduri, D. Melicher, R. Shay, B. Ur, L. Bauer, N. Christin, L. F. Cranor, and M. L. Mazurek. Diversify to survive: Making passwords stronger with adaptive policies. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS)*, pages 1–12, Santa Clara, CA, USA, 2017. USENIX Association. ISBN 978-1-931971-39-3. URL <https://www.usenix.org/conference/soups2017/technical-sessions/presentation/segreti>.
- [156] A. Senarath and N. A. G. Arachchilage. Understanding software developers’ approach towards implementing data minimization. *arXiv preprint arXiv:1808.01479*, 2018.
- [157] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor. Can long passwords be secure and usable? In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems, CHI '14*, pages 2927–2936, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557377. URL <http://doi.acm.org/10.1145/2556288.2557377>.
- [158] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor. Designing password policies for strength and usability. *ACM Transactions on Information and System Security (TISSEC)*, 18(4):13:1–13:34, May 2016. ISSN 1094-9224. doi: 10.1145/2891411. URL <http://doi.acm.org/10.1145/2891411>.
- [159] S. Sheth, G. Kaiser, and W. Maalej. Us and them: A study of privacy requirements across north america, asia, and europe. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, pages 859–870, 2014. doi: 10.1145/2568225.2568244.
-

-
- [160] V. J. Shute. Who is likely to acquire programming skills? 1991.
- [161] D. I. Sjöberg, B. Anda, E. Arisholm, T. Dyba, M. Jorgensen, A. Karahasanovic, E. F. Koren, and M. Vokác. Conducting realistic experiments in software engineering. In *Proceedings international symposium on empirical software engineering*, pages 17–26, Piscataway, NJ, USA, 2002. IEEE, IEEE Press.
- [162] J. Smith, B. Johnson, E. Murphy-Hill, B.-T. Chu, and H. Richter. How developers diagnose potential security vulnerabilities with a static analysis tool. *IEEE Transactions on Software Engineering*, 2018.
- [163] L. Sousa, R. Oliveira, A. Garcia, J. Lee, T. Conte, W. Oizumi, R. de Mello, A. Lopes, N. Valentim, E. Oliveira, et al. How do software developers identify design problems? a qualitative analysis. In *Proceedings of the 31st Brazilian Symposium on Software Engineering (SBES'17)*, pages 54–63, 2017. doi: 10.1145/3131151.3131168.
- [164] D. Spadini, G. Çalikli, and A. Bacchelli. Primers or reminders? the effects of existing review comments on code review. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE'20)*, 2020. doi: 10.1145/3377811.3380385.
- [165] E. Stobert and R. Biddle. The password life cycle: User behaviour in managing passwords. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 243–255, Menlo Park, CA, USA, 2014. USENIX Association. ISBN 978-1-931971-13-3. URL <https://www.usenix.org/conference/soups2014/proceedings/presentation/stobert>.
- [166] C. Stransky, Y. Acar, D. C. Nguyen, D. Wermke, D. Kim, E. M. Redmiles, M. Backes, S. Garfinkel, M. L. Mazurek, and S. Fahl. Lessons learned from using an online platform to conduct large-scale, online controlled security experiments with software developers. In *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET'17)*, 2017.
- [167] A. Studio. Android studio and sdk tools android developers. <https://developer.android.com/studio/>, 2018. Accessed: 28-01-19.
- [168] J. Stylos and B. A. Myers. The implications of method placement on api learnability. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 105–112. ACM, 2008.
- [169] J. Sunshine, S. Egelman, H. Almuhiemedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In *Proceedings of the 18th Conference on USENIX Security Symposium, SSYM'09*, pages 399–416, Berkeley, CA, USA, 2009. USENIX Association. URL <http://dl.acm.org/citation.cfm?id=1855768.1855793>.
- [170] M. Tahaei and K. Vaniea. A survey on developer-centred security. *2019 IEEE European Symposium on Security and Privacy Workshops*, pages 129–138, 2019.
- [171] D. R. Thomas. A general inductive approach for analyzing qualitative evaluation data. *American journal of evaluation*, 27(2):237–246, 2006.
-

-
- [172] T. W. Thomas, M. Tabassum, B. Chu, and H. Lipford. Security during application development: An application security expert perspective. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.
- [173] A. M. Turk, 2018. URL www.mturk.com.
- [174] S. Türpe, L. Kocksch, and A. Poller. Penetration tests a turning point in security practices? organizational challenges and implications in a software development team. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.
- [175] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. L. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, and J. López. Helping users create better passwords. *USENIX*, 37(6):51–57, 2012. URL <http://www.ece.cmu.edu/~lbauer/papers/2012/login2012-passwords.pdf>.
- [176] B. Ur, F. Noma, J. Bees, S. M. Segreti, R. Shay, L. Bauer, N. Christin, and L. F. Cranor. I added ‘!’ at the end to make it secure: Observing password creation in the lab. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 123–140, Ottawa, 2015. USENIX Association. ISBN 978-1-931971-249. URL <https://www.usenix.org/conference/soups2015/proceedings/presentation/ur>.
- [177] B. Ur, J. Bees, S. M. Segreti, L. Bauer, N. Christin, and L. F. Cranor. Do users’ perceptions of password security match reality? In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI ’16, pages 3748–3760, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3362-7. doi: 10.1145/2858036.2858546. URL <http://doi.acm.org/10.1145/2858036.2858546>.
- [178] A. Vance. If your password is 123456, just make it hackme. *The New York Times*, 20, 2010.
- [179] D. Votipka, D. Abrokwa, and M. L. Mazurek. Building and validating a scale for secure software development self-efficacy. In *Proceedings of the 2020 Conference on Human Factors in Computing Systems (CHI’20)*, pages 1–20, 2020. doi: 10.1145/3313831.3376754.
- [180] R. Wash, E. Rader, R. Berman, and Z. Wellmer. Understanding password choices: How frequently entered passwords are re-used across websites. In *Twelfth Symposium on Usable Privacy and Security (SOUPS)*, pages 175–188, Denver, CO, 2016. USENIX Association. ISBN 978-1-931971-31-7. URL <https://www.usenix.org/conference/soups2016/technical-sessions/presentation/wash>.
- [181] R. Wash, E. Rader, and C. Fennell. Can people self-report security accurately?: Agreement between self-report and behavioral measures. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI ’17, pages 2228–2232, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3025911. URL <http://doi.acm.org/10.1145/3025453.3025911>.
- [182] J. Weinberger and A. P. Felt. 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>, 2016.
-

-
- [183] C. Weir, A. Rashid, and J. Noble. How to improve the security skills of mobile app developers? comparing and contrasting expert views. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.
- [184] T. Whalen and K. M. Inkpen. Gathering evidence: use of visual security cues in web browsers. In *Proceedings of Graphics Interface 2005*, pages 137–144. Canadian Human-Computer Communications Society, 2005.
- [185] C. Wijayarathna and N. A. Arachchilage. Am i responsible for end-user’s security? Baltimore, MD. USENIX Association. URL <https://wsiw2018.l3s.uni-hannover.de/>.
- [186] C. Wijayarathna and N. A. G. Arachchilage. Why johnny can’t store passwords securely?: A usability evaluation of bouncycastle password hashing. In *Proceedings of the 22Nd International Conference on Evaluation and Assessment in Software Engineering 2018, EASE’18*, pages 205–210, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-6403-4. doi: 10.1145/3210459.3210483. URL <http://doi.acm.org/10.1145/3210459.3210483>.
- [187] A. C. Williams, H. Kaur, S. Iqbal, R. W. White, J. Teevan, and A. Fourney. Mercury: Empowering programmers’ mobile work practices with microproductivity. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST’19)*, pages 81–94, 2019. doi: 10.1145/3332165.3347932.
- [188] J. Witschey, O. Zielinska, A. Welk, E. Murphy-Hill, C. Mayhorn, and T. Zimmermann. Quantifying developers’ adoption of security tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE’15)*, pages 260–271, 2015. doi: 10.1145/2786805.2786816.
- [189] M. S. Wogalter, V. C. Conzola, and T. L. Smith-Jackson. Research-based guidelines for warning design and evaluation. *Applied Ergonomics*, 33(3):219 – 230, 2002. ISSN 0003-6870. doi: [https://doi.org/10.1016/S0003-6870\(02\)00009-1](https://doi.org/10.1016/S0003-6870(02)00009-1). URL <http://www.sciencedirect.com/science/article/pii/S0003687002000091>. Fundamental Reviews in Applied Ergonomics 2002.
- [190] G. Wurster and P. C. van Oorschot. The developer is the enemy. In *Proceedings of the 2008 New Security Paradigms Workshop*, pages 89–97, New York, NY, USA, 2009. ACM, ACM.
- [191] J. Xie, H. Lipford, and B.-T. Chu. Evaluating interactive support for secure programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2707–2716, 2012.
- [192] K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith. Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 158–177, Piscataway, NJ, USA, 2016. IEEE, IEEE.
- [193] A. Yamashita and L. Moonen. Do developers care about code smells? an exploratory survey. In *2013 20th Working Conference on Reverse Engineering (WCRE’13)*, pages 242–251. IEEE, IEEE, 2013. doi: 10.1109/WCRE.2013.6671299.
-

- [194] A. Yamashita and L. Moonen. Surveying developer knowledge and interest in code smells through online freelance marketplaces. In *User Evaluations for Software Engineering Researchers (USER), 2013 2nd International Workshop on*, pages 5–8, San Francisco, CA, USA, 2013. IEEE, IEEE. doi: 10.1109/USER.2013.6603077.
-