# Do you *really* code? Designing and Evaluating Screening Questions for Online Surveys with Programmers

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

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

Stefan Horstmann
*University of Bonn*
Bonn, Germany
Stefan.Horstmann@gmx.net

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



***Abstract*—Recruiting professional programmers in sufficient numbers for research studies can be challenging because they often cannot spare the time, or due to their geographical distribution and potentially the cost involved. Online platforms such as Clickworker or Qualtrics do provide options to recruit participants with programming skill; however, misunderstandings and fraud can be an issue. This can result in participants without programming skill taking part in studies and surveys. If these participants are not detected, they can cause detrimental noise in the survey data. In this paper, we develop screener questions that are easy and quick to answer for people with programming skill but difficult to answer correctly for those without. In order to evaluate our questionnaire for efficacy and efficiency, we recruited several batches of participants with and without programming skill and tested the questions. In our batch 42% of Clickworkers stating that they have programming skill did not meet our criteria and we would recommend filtering these from studies. We also evaluated the questions in an adversarial setting. We conclude with a set of recommended questions which researchers can use to recruit participants with programming skill from online platforms.**

## I. Introduction

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., [1], [2], [3], [4], [5], [6]). Unfortunately, finding enough willing participants locally can be difficult; Naiakshina et al. [2] 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., [7], [8], [9], [10], [11], [12], [13], [14], [15]). 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 [13], [16], [17]. However, since researchers often offer significantly higher compensation than for end-user studies [1], [2], [11], 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 [18], [19]. For *programming skill*, we use the definition of Bergersen et al. [19], which is in accordance to the definition used in psychology [18], [20], [21], [22], [23]: "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., [3], [7], [9], [10], [15], [19], [24], [25], [26], [27]). 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 [13], [14], [16], [17], [28], [29], [30], [31], [27], [32], [33], [34], [25].

Researchers can also use online recruitment platforms such as Clickworker [35] or Qualtrics [36] for developer studies (e.g., [13], [29]). 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. Danilova et al. [29] came across such a case. They 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, the authors presented participants with some pseudocode that printed "hello world" backwards and asked what the output would be. The authors reported that 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, the authors 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.

## II. Related Work

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.

### A. 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 [7], [9], [10], [37], [24] or on freelance platforms for developers [14], [38], [12], [11]. For example, when investigating the effect of happiness on productivity, Graziotin et al. [37] recruited their participants by contacting software developers on GitHub [39]. GitHub developers were also recruited for a security developer study conducted by Acar et al. [10] to investigate how they perform with regard to security-related tasks. Furthermore, Meyer et al. [40], 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 [41]. 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 [42], 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 [38], 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.

### B. Identifying Programming Skill

Bergersen et al. [19] 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. [43] 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. [19] 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 [25], 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 [3], 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.

Danilova et al. [29] 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 Introduction. To further evaluate the used question, we included it to our question set as Q16.

Assal and Chiasson [13] 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 where 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.

## III. Methodology

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

### A. Instrument Requirements

Unlike Bergersen et al.'s [19] 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

TABLE I: 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.OutOfBound | 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 |

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 paper.

### *B. Survey*

We used Dillman's pre-testing process to develop an online survey with 16 different questions [44, 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 I. The full questions can be found in the supplementary material. 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 [45]. 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 the study by Danilova et al. [29].

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 [46] 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 [47].

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 VII-B). 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 supplementary material. 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.

### C. 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) [48, 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 [49], [50], [51]. 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).

### D. 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.

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.

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.

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

TABLE II: Demographics of the participants (n = 249)

| Group | Sample | n | Gender | Age | Country of Residence | General Programming Experience [years] |
|---|---|---|---|---|---|---|
| **Programmer** | *CS students* | 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 |
| | *Professional developers* | 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** | *Econ students* | 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 |
| | *Clickworkers without programming skill* | 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** | *Clickworkers with programming skill* | 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** | *Clickworkers without programming skill* | 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 supplementary material for further details on occupation and country of residence.

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 II. 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).

## IV. 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.

## V. 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.

## VI. 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 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.

### A. 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 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

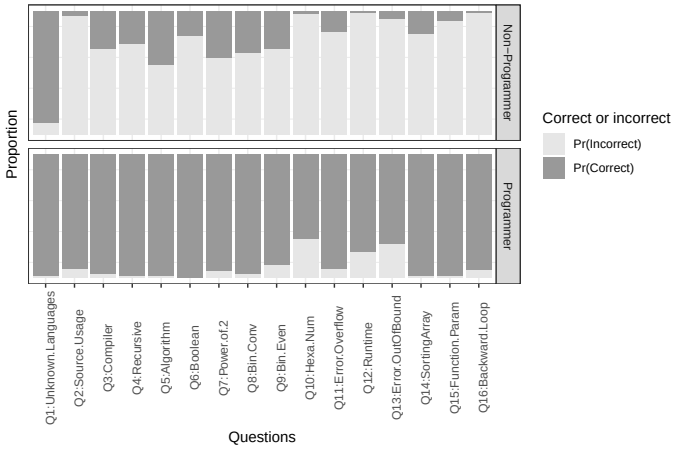


Fig. 1: Plot of latent class analysis with the programmer and non-programmer groups.

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.

*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 supplementary material.

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 ErrorOutOfBound (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.

### B. 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 VII-B 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 supplementary material.

## VII. 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.

### A. 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.

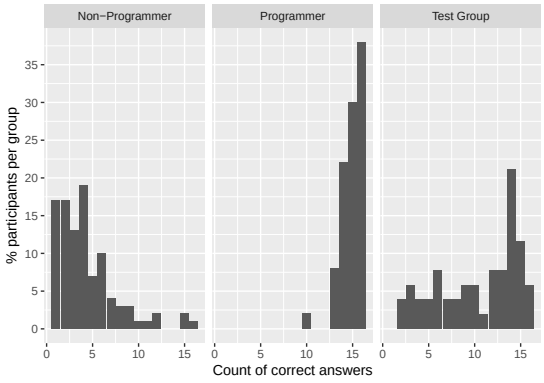


Fig. 2: Number of correct solutions of all 16 questions per group.

**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."

*Comparison of the Groups:* Table III 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 2.

### *B. 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 III). 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 supplementary material for more details).

TABLE III: 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.Language | 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.

**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 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 3 shows the time taken to solve the questions correctly for the programmer and the attack groups. We excluded Q1

TABLE IV: 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

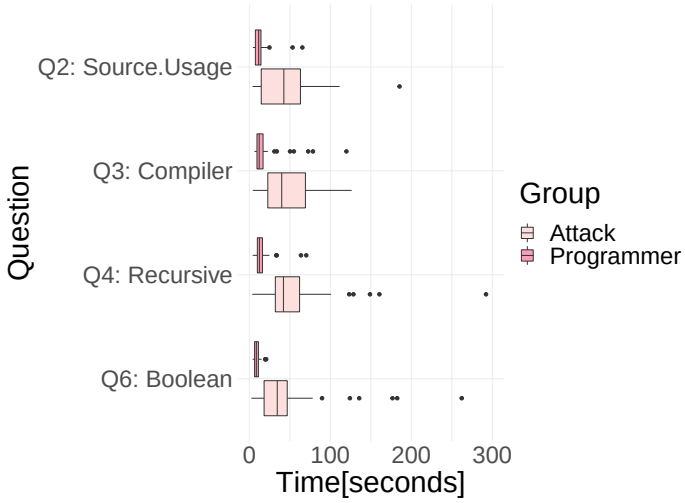


Fig. 3: Comparison knowledge questions: Time to solve each question **correctly**.

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.

## VIII. 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 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 IV 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 supplementary material, we provide details of how we chose these time limits. Figure 4 shows the distribution over the 6 recommended questions of correct solutions in the groups with the time limits applied.

## IX. Conclusion

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.

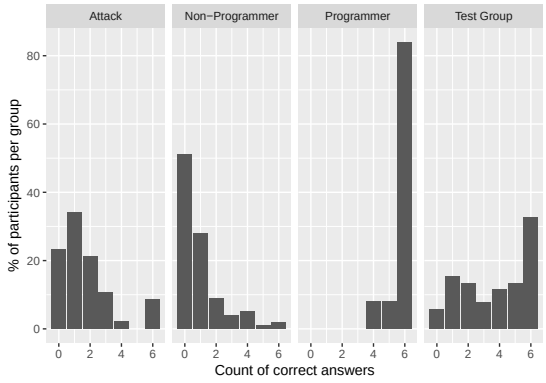


Fig. 4: Distribution of correct solutions of all 6 recommended questions per group with time limits applied as in Table IV.

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.

## Acknowledgments

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

## References

[1] 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*, ser. CCS'17. New York, NY, USA: ACM, 2017, pp. 311–328. [Online]. Available: https://doi.acm.org/10.1145/3133956.3134082

[2] 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)*. Baltimore, MD: USENIX Association, 2018, pp. 297–313. [Online]. Available: https://www.usenix.org/conference/soups2018/presentation/naiakshina

[3] 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'16)*, 2016, pp. 289–305. [Online]. Available: https://doi.org/10.1109/SP.2016.25

[4] 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)*. IEEE, 2017, pp. 94–105.

[5] 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)*, 2019, pp. 81–94.

[6] 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)*, 2016, pp. 3192–3203.

[7] 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*, ser. EASE'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 205–210. [Online]. Available: https://doi.org/10.1145/3210459.3210483

[8] 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)*, 2017, pp. 54–63.

[9] 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'18)*, 2018, pp. 265–281.

[10] 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'17)*, 2017, pp. 81–95.

[11] 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 Conference on Human Factors in Computing Systems*, ser. CHI'19. New York, NY, USA: ACM, 2019, pp. 140:1–140:12. [Online]. Available: https://doi.acm.org/10.1145/3290605.3300370

[12] 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.

[13] 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*, ser. CHI'19. New York, NY, USA: ACM, 2019, pp. 289:1–289:13. [Online]. Available: https://doi.acm.org/10.1145/3290605.3300519

[14] A. Yamashita and L. Moonen, "Do Developers Care about Code Smells? An Exploratory Survey," in *2013 20th Working Conference on Reverse Engineering (WCRE'13)*. IEEE, 2013, pp. 242–251.

[15] 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)*, 2020, pp. 1–13.

[16] 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)*, 2018, pp. 572–583.

[17] 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.

[18] P. M. Fitts and M. I. Posner, "Human performance." 1967.

[19] 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)*, vol. 40, no. 12, pp. 1163–1184, 2014.

[20] V. J. Shute, "Who is Likely to Acquire Programming Skills?" 1991.

[21] J. R. Anderson, "Skill Acquisition: Compilation of Weak-Method Problem Situations." *Psychological review*, vol. 94, no. 2, p. 192, 1987.

[22] J. R. Anderson, F. G. Conrad, and A. T. Corbett, "Skill Acquisition and the LISP Tutor," *Cognitive Science*, vol. 13, no. 4, pp. 467–505, 1989.

[23] J. R. Anderson, R. Farrell, and R. Sauers, "Learning to Program in LISP," *Cognitive Science*, vol. 8, no. 2, pp. 87–129, 1984.

[24] 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'17)*. IEEE, 2017, pp. 154–171.

[25] R. Balebako, A. Marsh, J. Lin, J. I. Hong, and L. F. Cranor, "The Privacy and Security Behaviors of Smartphone App Developers," 2014.

[26] L. Prechelt, "Plat_Forms: A Web Development Platform Comparison by an Exploratory Experiment Searching for Emergent Platform Properties," *IEEE Transactions on Software Engineering*, vol. 37, no. 1, pp. 95–108, 2011.

[27] 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*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 935–946. [Online]. Available: https://doi.org/10.1145/2884781.2884790

[28] 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)*, 2020, pp. 1–20.

[29] 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.

[30] 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)*, 2015, pp. 260–271.

[31] 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)*, 2014, pp. 859–870.

[32] 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*, vol. 17, no. 04, 2017.

[33] 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)*, 2015, pp. 239–254.

[34] 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*, ser. SOUPS'18. USA: USENIX Association, 2018, p. 315–328.

[35] "Clickworker," https://www.clickworker.de/, Accessed: January 2020. [Online]. Available: https://www.clickworker.de/

[36] "Qualtrics," https://www.qualtrics.com, Accessed: January 2020. [Online]. Available: https://www.qualtrics.com

[37] 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)*. IEEE, 2017, pp. 362–364.

[38] A. Yamashita and L. Moonen, "Surveying Developer Knowledge and Interest in Code Smells through Online Freelance Marketplaces," in *2013 2nd International Workshop on User Evaluations for Software Engineering Researchers (USER'13)*. IEEE, 2013, pp. 5–8.

[39] "Github," https://github.com/, Accessed: January 2020. [Online]. Available: https://github.com/

[40] 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.

[41] 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 (ESEM'16)*, 2016, pp. 1–6.

[42] "Ghtorrent," https://ghtorrent.org/, Accessed: January 2020. [Online]. Available: https://ghtorrent.org/

[43] 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)*. IEEE, 2012, pp. 73–82.

[44] 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.

[45] 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)*, May 2017, pp. 121–136.

[46] F. Y. Kung, N. Kwok, and D. J. Brown, "Are Attention Check Questions a Threat to Scale Validity?" *Applied Psychology*, vol. 67, no. 2, pp. 264–283, 2018.

[47] P. J. Lavrakas, *Encyclopedia of survey research methods*. Sage Publications, 2008.

[48] A. Field, J. Miles, and Z. Field, *Discovering statistics using R*. Sage publications, 2012.

[49] A. L. McCutcheon, *Latent class analysis*. Sage, 1987, no. 64.

[50] L. M. Collins and S. T. Lanza, *Latent class and latent transition analysis: With applications in the social, behavioral, and health sciences*. John Wiley & Sons, 2009, vol. 718.

[51] 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.'11)*, vol. 42, no. 10, pp. 1–29, 2011.