Testing Time Limits in Screener Questions for Online Surveys with Programmers

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

Stefan Horstmann
University of Bonn
Stefan.Horstmann@gmx.net

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

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

ABSTRACT

Recruiting study participants with programming skill is essential for researchers. As programming is not a common skill, recruiting programmers as participants in large numbers is challenging. Platforms like Amazon MTurk or Qualtrics offer to recruit participants with programming knowledge. As this is self-reported, participants without programming experience could still take part, either due to a misunderstanding or to obtain the study compensation. If these participants are not detected, the data quality will suffer. To tackle this, Danilova et al. [11] developed and tested screening tasks to detect non-programmers. Unfortunately, the most reliable screeners were also those that took the most time. Since screeners should take as little time as possible, we examine whether the introduction of time limits allows us to create more efficient (i.e., quicker but still reliable) screeners. Our results show that this is possible and we extend the pool of screeners and make recommendations on how to improve the process.

CCS CONCEPTS

• **Human-centered computing → Empirical studies in HCI**.

KEYWORDS

Developer Study; Methodology Developer Studies

ACM Reference Format:
Anastasia Danilova, Stefan Horstmann, Matthew Smith, and Alena Naiakshina. 2022. Testing Time Limits in Screener Questions for Online Surveys with Programmers. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3510003.3510223

1 INTRODUCTION

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 [10, 24, 26–28]. Researchers have been recruiting programmers from known IT-firms [22, 23], local programming groups [2] or GitHub [1, 3, 18, 20, 31].

Some online services like Clickworker [9], Qualtrics [29] or Amazon MTurk [30] 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 by Danilova et al. in [11]. They tested the tasks with programmers and non-programmers. As "programmer," we refer to participants as defined by Danilova et al. [11]: participants with programming skills from previous studies and participants who have worked with imperative programming languages (e.g., Java, C, Python). 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. Danilova et al. 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. Therefore, we investigated the research question: *Can the effectiveness of screening questions for non-programmers be improved by introducing time constraints?*

In this paper, 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 Danilova et al. [11]. We explored the influence of different time constraints on the correctness rate for programmers and non-programmers in an

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9221-1/22/05.
https://doi.org/10.1145/3510003.3510223

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 of Danilova et al. [11]. Finally, we compared the success rate and the required time to solve the tasks with programmers in a countdown 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. Danilova et al. [12] 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 [11] for the code comprehension question), it would yield an additional cost of 100*2.87*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).

2 RELATED WORK

This section looks at the different recruitment options researchers used in previous studies with software developers. Secondly, we present various studies where participants were filtered based on their software development skills. Lastly, we discuss the work done by Danilova et al. [11] in greater detail as this paper builds on their results.

2.1 Recruitment

In many cases, researchers aimed to target only software developers during the recruitment phase of their studies. One frequently used site is GitHub [16], which is generally used to contact developers using the services GitHub provides [1, 3, 18, 20, 31]. For example, Graziotin et al. [20] contacted developers on GitHub and recruited them for a study concerning the influence of worker happiness on productivity. Additionally, Acar et al. [3] recruited GitHub participants for a study concerning developers' performance regarding to security-related tasks.

Other researchers contacted companies and recruited employees as participants. When Meyer et al. [22] did a study on job satisfaction and productivity, they recruited all participants from Microsoft as to ensure that only software developers took part in the study. Furthermore, in [23], Murhpy-Hill et al. recruited participants from three different companies for a study about software developers' productivity.

Furthermore, Baltes and Diehl [6] compared different recruitment methods. They examined the recruitment of participants, using a personal network of contacts, online communities and networks, reaching out to companies, public media as well as advertising the survey by software developers and GHTorrent [15], where data on public GitHub projects is collected. They recommended using public media and advertising the survey by other software engineers as the most efficient strategies. They additionally proposed to use crowdsourcing platforms (e.g., Clickworker [9], Amazon Mechanical Turk [30]) and commercial recruitment services, but also mentioned concerns regarding the suitability of those services for the recruitment of software developers.

Another approach used for the recruitment of developers was the usage of freelance platforms for studies [7, 25, 32, 33]. However, Yamashita and Moonen [33] expressed concerns regarding the recruitment on the freelance platform Freelance.com [14] as all skills are self-reported by the members. They, therefore, suggested the use of skill tests to validate research findings.

2.2 Programming skills

An instrument to evaluate the participants' Java programming skills was created and evaluated by Bergersen et al. [8], using a set of 19 programming tasks. The purpose of the instrument was to aid recruiters in finding the most skilled programmers in a group of job applicants. They conducted a two-day study including 65 professional Java developers. However, time limits for the tasks ranged from 10 to 45 minutes per task, making them too long to include in an online survey and thus are not suitable for filtering out participants quickly.

Feigenspan et al. [13] used program comprehension tasks to analyze the self-reported programming skills of 125 students. In a survey taking 40 minutes, they found a correlation between the participants' performance and their self-reported level of experience. While some program comprehension tasks were used in this paper, the goal was to identify participants without any programming skills instead of measuring the skill level.

Balebako et al. [5] examined the security behavior of app developers. At first, participants for a semi-structured interview were recruited through online postings, app developer meet-ups, and the researchers' social network. All participants had to fill out a screening survey before participating, with two questions designed to verify the participants' technical knowledge. In an online follow-up survey, participants were recruited in different online forums. Participants were required to pass knowledge and attention checks. After the screening, 232 of 460 responses had to be discarded. One of the tasks used by the authors to filter the participants was to ask participants to name an IDE they knew or had previously worked with. The researchers then validated their answers, which they noted to be a difficult task due to many existing IDEs. Based on this, two multiple-choice questions asking for used and known programming languages were included in this paper, with the added difference that a number of fake programming languages was included in the answer possibilities.

Acar et al. [2] conducted an online survey and a lab study concerning the effect of information resources on security during software development. Approximately 50.000 developers, whose contact information was found on GooglePlay [17], were sent a recruitment mail. Seven out of the 302 participants who completed the survey had to be excluded due to invalid answers. No additional methods to filter non-developers were applied. When recruiting for the lab study, Acar et al. targeted participants who had at least

one year of app development experience or had completed a course on Android development. They contacted participants via online advertisements, emails to local and online groups, and university students who studied related subjects. At first, participants had to complete a programming task to participate. After participants complained about the time it took to complete the programming task, it was replaced with five multiple-choice questions, three of which had to be solved correctly. This may indicate that programming tasks are not optimal to filter participants in online surveys. The multiple-choice tasks dealt with the subject of app development, and thus did not suit this paper as the goal is to include all software developers, not just those skilled in app development. However, some similar themes were included in the question set, such as error handling.

Assal and Chiasson [4] recruited participants for an online survey on Qualtrics regarding developers' software security process. To ensure a baseline of understanding, the authors showed participants different software security descriptions and prevented them from processing until they had selected the correct one. Only participants providing invalid data were excluded from the analysis.

In [12], Danilova et al. explored software developers' preferences regarding IDE warnings in a lab study and an online survey. They recruited developers via Qualtrics [29] for the online survey and included a pseudo-code program printing the words "hello world" backward. Participants were tasked with selecting the output in a multiple-choice question, which only 33 of 129 participants correctly solved. The task was included in this paper.

2.3 Baseline study

This paper expands on the work by Danilova et al. [11], 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 [9] 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, Danilova et al. 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.

3 METHODOLOGY

In this section, we describe the methodology for the three conducted surveys in this paper, 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 3.1. Secondly, we discuss our study conditions in Section 3.2. Thirdly, we tested the questions with computer science students with different time constraints as described in Section 3.3.1. Finally, we simulated an attack scenario with Clickworker participants as described in Section 3.3.2. An attention check question [21] was placed randomly during the survey to filter out careless respondents (see Supplementary Material). After solving all the tasks, participants were asked if they had felt under pressure during the survey, and demographic information was collected (see Supplementary Material).

3.1 Tasks

Our potential set of filter tasks consists of already established tasks from [11] and new tasks, which we developed to extend the question pool. A list of all questions used can be found in Table 1, with the first six being the recommended tasks from Danilova et al. [11] and the newly developed tasks. The detailed questions with all the answer possibilities are additionally listed in the supplementary material.

Baseline questions. Multiple questions (**Boolean**, **Compiler**, **Recursive**, **Backward.Loop**, **Sorting.Array** and **Source.Usage**) from Danilova et al.'s 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 answers, they may still be used as a filter based on the time to solve the tasks.

New questions. Based on the results of the previous surveys done by Danilova et al. [11], a new set of questions was developed. Danilova et al. 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 **Backwards.Loop** 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

Table 1: Overview of all questions used for the timed scenarios

	Baseline Questions from [11]	**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? (0-based indexing)	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

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. [5]. 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.

3.2 Conditions

All conditions are summarized in Table 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.

3.3 Participants

To calculate the number of participants required to create statistically significant results, a power analysis was conducted with G*power [19]. 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 conducted by Danilova et al. [11], 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 3.

3.3.1 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 supplementary material.

Table 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

Table 3: Demographics of the Participants (n = 98)

	Programmer (n = 74)			*Non-Programmer (n = 24)*
	Base (n = 23)	**No Limit (n = 26)**	**Countdown (n = 25)**	**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

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 3.3). 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.

3.3.2 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 [9] 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 supplementary material.

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 the payments by Danilova et al. in [11]. With ten tasks providing bonus payment, this resulted in a maximum of 22€ per participant.

3.4 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.

3.5 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.

3.6 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 [9] 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.

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.

This instrument can be applied in software engineering research, where researchers otherwise would have to rely on self-reported programming skills (snowball sampling, sampling over forums, or online recruiting services like Qualtrics, LimeSurvey, QuestionPro, etc.). However, Google forms offers time limits for questionnaires but not single questions. SurveyMonkey does not offer a time limit at the moment.

4 RESULTS

In this section, the studies with programmers and non-programmers are discussed. First, the tasks introduced in Table 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 [11]. 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.

4.1 Programmers

A summary of the time the participants required per task, and the percentage of correct answers is available in the supplementary material. 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.

4.1.1 Ensuring comparability with the baseline study. First, it had to be ensured that participants were similar in skill level to the programmers previously recruited by Danilova et al. in the baseline study [11] 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 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 nor 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 the baseline study of Danilova et al.

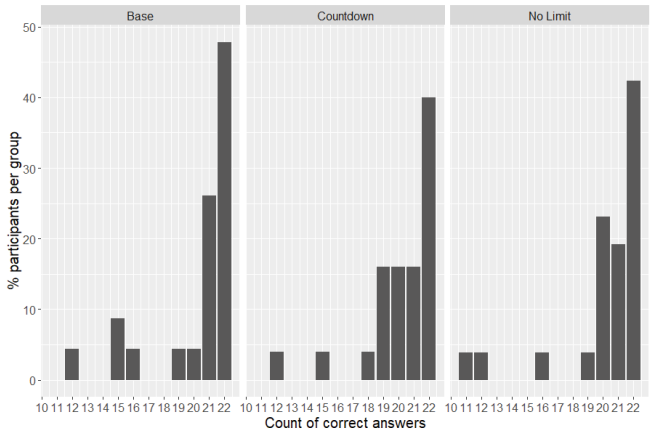

Figure 1: Number of correct solution by each programmer group

4.1.2 Comparison between groups. Figure 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

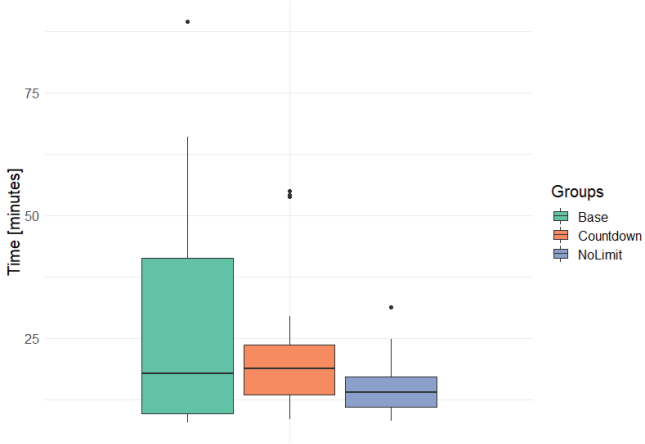

Figure 2: Time required to complete the survey by each programmer group

not show a significant difference between the groups in any of the tasks.

In Figure 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).

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.

4.1.3 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.

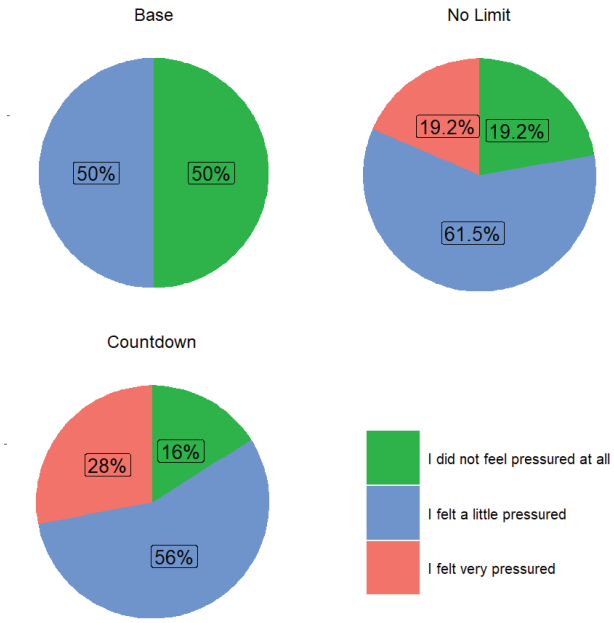

Figure 3: Perceived pressure by each programmer group
Limitation: Since this question was added later to the survey, it was only shown to ten participants of the Base group. All other groups had the question already included in their survey.

4.1.4 Pressure. 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 3.

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

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

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.

4.2 Non-programmers - Timed attack scenario

The attack scenario from Danilova et al. [11] 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.

4.2.1 Comparison to the baseline attack scenario. As participants in the attack scenario used by Danilova et al. [11] 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.

4.2.2 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 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.

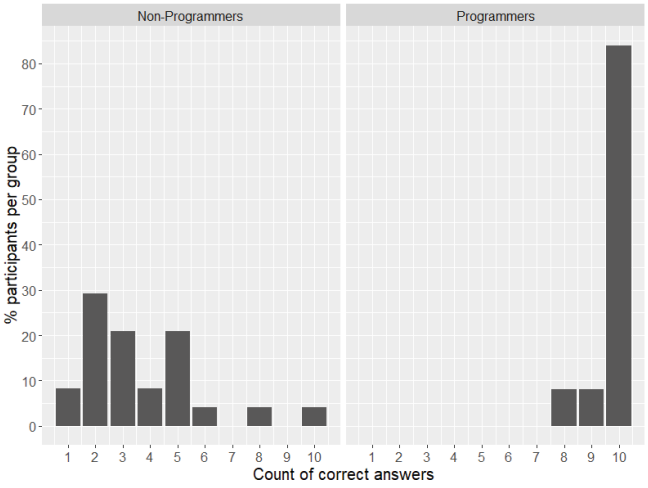

Figure 4: Number of correct solutions by programmers and non-programmers

The results for both groups are presented in Table 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.

4.2.3 Aid used. The non-programmer participants were asked if they used any form of outside help to solve the tasks. Possible

Table 4: Comparison of programmers and attack scenario

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

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 conducted by Danilova et al. in [11]. 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 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 [11]. 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 [11] 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.

5.1 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. While it would have been beneficial to test the same sets of questions with programmers and non-programmers, we learned from the study with programmers that some questions could not be chosen for the instrument due to a high error rate and thus tested only a subset with non-programmers.

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.

Table 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 arr[7,3,5,1,9], what could the command arr[3] return? The array starts with 0.	1.5m	96%	16.67%

5.2 Time constraints

While the *No limit* group participants completed the entire survey the fastest, this condition might not apply to non-programmers because they still would have the time to google the solutions for the knowledge-based questions. To give no limit yet cut them off implicitly after some time might be a cheap way, but it is technically challenging and raise ethical concerns. Thus, the *No limit* condition might increase questionnaires' time, resulting in potentially higher research costs than *Countdown*.

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.

5.3 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 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.

6 CONCLUSION

To ensure data quality, researchers should screen out non-programmers from online surveys requiring programming skills. In this paper, we tested existing screener questions suggested by Danilova et al. in [11] 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.

ACKNOWLEDGMENTS

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

REFERENCES

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

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2016. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Jose, CA, USA, 289–305.

[3] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L Mazurek, and Sascha Fahl. 2017. Security developer studies with github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. USENIX Association, Santa Clara, CA, USA, 81–95.

[4] Hala Assal and Sonia Chiasson. 2019. 'Think secure from the beginning' A Survey with Software Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow, UK, 1–13.

[5] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason I Hong, and Lorrie Faith Cranor. 2014. The privacy and security behaviors of smartphone app developers. (2014).

[6] Sebastian Baltes and Stephan Diehl. 2016. Worse than spam: Issues in sampling software developers. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, Ciudad Real, Spain, 1–6.

[7] Jason Bau, Frank Wang, Elie Bursztein, Patrick Mutchler, and John C Mitchell. 2012. Vulnerability factors in new web applications: Audit tools, developer selection & languages. *Stanford, Tech. Rep* (2012).

[8] Gunnar R Bergersen, Dag IK Sjøberg, and Tore Dybå. 2014. Construction and validation of an instrument for measuring programming skill. *IEEE Transactions on Software Engineering* 40, 12 (2014), 1163–1184.

[9] Clickworker. 2021. Retrieved August 16, 2021 from https://www.clickworker.com/ https://www.clickworker.com/, Accessed: September 2020.

[10] Anastasia Danilova, Alena Naiakshina, Johanna Deuter, and Matthew Smith. 2020. 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)*. USENIX Association, Boston, MA, USA, 165–183. https://www.usenix.org/conference/soups2020/presentation/danilova

[11] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. 2021. Do you really code? Designing and Evaluating Screening Questions for Online Surveys with Programmers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, Madrid, Spain, 537–548.

[12] Anastasia Danilova, Alena Naiakshina, and Matthew Smith. 2020. One size does not fit all: a grounded theory and online survey study of developer preferences for security warning types. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, Seoul, South Korea, 136–148.

[13] Janet Feigenspan, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. 2012. Measuring programming experience. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. IEEE, Passau, Bavaria, Germany, 73–82.

[14] Freelancer. 2021. Retrieved August 16, 2021 from https://www.freelancer.com/ freelancer.com, Accessed: September 2020.

[15] GHTorrent. 2021. Retrieved August 16, 2021 from https://ghtorrent.org/

[16] GitHub. 2021. Retrieved August 16, 2021 from https://github.com/

[17] GooglePlay. 2021. Retrieved August 16, 2021 from https://play.google.com

[18] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. 2018. Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic {API} Misuse. In *Fourteenth Symposium on Usable Privacy and Security ({SOUPS} 2018)*. USENIX Association, Baltimore, MD, USA, 265–281.

[19] G*Power. 2021. Retrieved August 16, 2021 from gpower.hhu.de

[20] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. 2017. 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)*. IEEE, Buenos Aires, Argentina, 362–364.

[21] Franki YH Kung, Navio Kwok, and Douglas J Brown. 2018. Are Attention Check Questions a Threat to Scale Validity? *Applied Psychology* 67, 2 (2018), 264–283. https://doi.org/10.1111/apps.12108

[22] Andre Meyer, Earl T Barr, Christian Bird, and Thomas Zimmermann. 2019. Today was a good day: The daily life of software developers. *IEEE Transactions on Software Engineering* (2019).

[23] Emerson Murphy-Hill, Ciera Jaspan, Caitlin Sadowski, David C. Shepherd, Michael Phillips, Collin Winter, Andrea Knight Dolan, Edward K. Smith, and Matthew A. Jorde. 2019. What Predicts Software Developers' Productivity? *Transactions on Software Engineering* (2019).

[24] A. Naiakshina, A. Danilova, E. Gerlitz, and M. Smith. 2020. On Conducting Security Developer Studies with CS Students: Examining a Password Storage Study with CS Students, Freelancers, and Company Developers. In *ACM CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu, HI, USA, 1–13. https://doi.org/10.1145/3313831.3376791

[25] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. 2019. " If you want, I can store the encrypted password" A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow, Scotland, Uk, 1–12.

[26] A. Naiakshina, A. Danilova, E. Gerlitz, E. von Zezschwitz, and M. Smith. 2019. "If you want, I can store the encrypted password." A Password-Storage Field Study with Freelance Developers. In *ACM CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow, Scotland, Uk, 1–12. https://doi.org/10.1145/3290605.3300370

[27] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, and Matthew Smith. 2017. Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study. In *24th ACM Conference on Computer and Communications Security - ACM CCS 2017*. ACM, Dallas, Texas, USA, 311–328.

[28] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, and Matthew Smith. 2018. Deception Task Design in Developer Password Studies: Exploring a Student Sample. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, USA, 297–313.

[29] Qualtrics. 2021. Retrieved August 16, 2021 from https://www.qualtrics.com

[30] Amazon Mechanical Turk. 2018. Retrieved August 16, 2021 from www.mturk.com

[31] Chamila Wijayarathna and Nalin AG Arachchilage. 2018. 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*. ACM, Christchurch, New Zealand, 205–210.

[32] Aiko Yamashita and Leon Moonen. 2013. Do developers care about code smells? An exploratory survey. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, Koblenz, Germany, 242–251.

[33] Aiko Yamashita and Leon Moonen. 2013. 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)*. IEEE, San Francisco, CA, USA, 5–8.