This is an author's version. The final publication will be published in the Software Quality Journal. The final publication is available at http://link.springer.com, DOI is 10.1007/s11219-016-9333-7

Onboarding Inexperienced Developers: Struggles and Perceptions Regarding Automated Testing

Raphael Pham · Stephan Kiesling · Leif Singer · Kurt Schneider

Abstract Previous research found that inexperienced software engineers may tend to view automatic testing as a waste of time and as an activity completely separate from programming. This could have a negative impact on their later careers, and could be a sign that improvements in software engineering education are needed when it comes to testing. At the same time, this stance could negatively influence the perception that practitioners have of recent university graduates.

To explore this issue, we conducted a qualitative study and surveyed 170 and interviewed 22 practitioners about their experiences with recent graduates, focusing on software testing skills. We find that practitioners do recognize a skill gap between university graduates and industry expectations, and that this perception could be engrained deeply enough already to influence hiring practices. Practitioners use different and at times costly strategies to alleviate this skill gap, such as training and mentoring efforts. We validated core findings in a survey with 698 professional software developers.

Raphael Pham
Welfengarten 1, 30163 Hanover, Germany
Tel.: +49-511-76219672
Fax: +49-511-76219679
E-mail: Raphael.Pham@inf.uni-hannover.de

Stephan Kiesling
Welfengarten 1, 30163 Hanover, Germany
Tel.: +49-511-76219680
Fax: +49-511-76219679
E-mail: Stephan.Kiesling@inf.uni-hannover.de

Leif Singer
University of Victoria, 3800 Finnerty Rd, Canada
E-mail: lsinger@uvic.ca

Kurt Schneider
Welfengarten 1, 30163 Hanover, Germany
Tel.: +49-511-76219666
Fax: +49-511-76219679
E-mail: Kurt.Schneider@inf.uni-hannover.de

Our qualitative insights can help industry, research, and educational institutions guide in-depth studies that explore the severity of the effects we have found. The coping strategies we have found can provide valuable starting points that can inform changes in how we educate the software engineers of the future.

Keywords Automated Testing · Onboarding · Newcomers · Software developer · Required skills

1 Introduction

Systematic software testing is an integral part of software engineering and abilities in this area are in high demand. Successful software companies promote a sophisticated software testing culture and award systematic testing the same importance as programming
[43]. However, practitioners have expressed concerns about the role of software testing in computer science education. Brechner, Director of Developer Training at Microsoft, identified five computer science education areas for improvement in order to meet industry needs, including high testability and use of automated tests [4]. University training in testing is described as "widely second-rate" [18] and in an Australian study about testing practices Ng et al. find that a surprisingly high number of practitioners are self-taught [27]. Similarly, Shepard et al. advocate a stronger focus on testing education and predict "a serious gap in the knowledge they need to be effective software developers" [35].

Besides teaching the core concepts, lecturers at universities try to lessen the blow of the 'shock of practice' that can happen when insufficiently prepared graduates enter work life. In this regard, Kaner et al. found that traditional lecturing left students unable to apply their knowledge in similar practical situations [23]. Current testing education is deemed too theoretical, with overly full curricula while testing is taught too late [13], giving opportunity to forming bad habits. This has spawned several new teaching approaches: Introducing testing as early as possible could induce a shift in how students behave and think about testing [22], [21], [11], [19], [26], [33]. Researchers report on different approaches to successfully introduce students to actually engage in testing practices instead of just learning all about them [3], [13], [12], [39], [25]. Moving away from theoretical examples, educators turn to open source projects [38], [6], [10], industry-oriented student projects [24] or cooperations with industry [5] to enrich students' experiences with real-world software development. It is, however, not clear whether these approaches have been widely adopted and whether or not they have been able to alleviate practitioners' concerns. What is the current extent of the 'Shock of Practice' regarding software testing? How do newcomers such as recent graduates fare in the real world?

The onboarding phase of an inexperienced developer is an important step in becoming a proficient software developer. If gaps in test knowledge and experience are not taken care of early on, the newcomer's professional progress

may be hindered. However, closing such gaps can be costly for employers as it encompasses additional phases of teaching. This study is a first step in understanding industry's view on the testing skills of new hires. While we are concerned with both industry and education, we place a stronger focus on industry's perspective on the matter.

While some of our insights may be perceived as well-known facts, we strive to present a in-depth and complete view of the current struggles and strategies of onboarding. This qualitative and comprehensive approach may provide companies struggling with similar problems an opportunity to find practical solutions.

2 Related Work

Recent Graduates as New Hires. In a previous study, we found that soon-to-be Bachelor graduates had considerable trouble applying testing techniques [29]. Systematic testing was often neglected for various reasons, and students expressed a dismissive attitude towards testing. Without guidance, these inexperienced developers misperceived the return of investment of test automation efforts.

Deak et al. underline the importance of proficient testing skills in graduates. Asking Norwegian computer science alumni about their job experiences, they find that proper testing skills are in very high demand [8]. They focus on ranking different engineering disciplines by practitioner need, but do not provide deeper qualitative insights into the required testing abilities. Similarly, Radermacher et al. [32] conducted a survey about common areas of struggle for recent graduates. Among other deficiencies, newcomers struggle with writing unit tests as well as proficiently using software tools. Radermacher et al. assume that a lack of problem solving abilities hinders newcomers in performing well. While Radermacher et al. provide a broader overview and describe multiple areas of struggle (such as 'soft skills', 'understanding of computer science concepts', 'use of software tools'), our study puts its focus on the testing abilities and deficiencies of newcomers and provides deeper insights into this topic. However, not all graduates struggle with testing. After shadowing eight newcomers at Microsoft, Begel et al. identify five areas of frequent problems [2]: cognition, collaboration, communication, orientation, and technical. Graduates showed good efforts in testing their application and wrote automated tests.

Onboarding in Open Source. Regarding onboarding of newcomers, open source projects have seen some attention. Steinmacher et al. provide a catalogue of barriers for onboarders of open source projects [40]. Here, newcomers are struggling with finding mentors for support. Similarly, Fagerholm et al. report that finding a suitable mentor improves performance of the newcomer [14].

While the onboarding process in open source often is observable, this is not generally true for commercial software companies. Our study focuses solely on such commercial onboarding processes. In this regard, Johnson et al. report on

onboarding at Google [20]. However, their study does not include information about the testing process.

Onboarding Strategies. Onboarding is generally easier for the newcomer when being technically supported, in open source development [7] as well as in commercial development. Begel et al. [1] advocate mentoring so that the mentor can act as a role model. In open source projects, mentorship can bring the newcomer up to speed faster but "requires a significant investment of time and effort" [15]. Similarly, de Marco concluded: "We all know that a new employee is quite useless on day one or even worse than useless, since someone else's time is required to begin bringing the new person up to speed." [9]. Sim et al. [36] consider it inefficient and disruptive, yet effective in passing on new information to the onboarder.

3 Study Design

In our study, we wanted to explore the views and concerns of practitioners who handle the onboarding phase of newly hired software engineers, with a focus on inexperienced newcomers such as recent university graduates. We had no prior knowledge about this population's views and employed a Grounded Theory approach [17]. We formulated our initial field of interest as two research questions:

RQ 1: Do software companies see problems with the testing skills of new hires?
RQ 2: What are these problems, exactly?

During our studies, we became more and more interested in *how* practitioners manage a lack of testing skills and added the following research questions.

RQ 3: How are companies coping with these problems?

RQ 4: What is the impact of new hires' lack of testing skills?

Insights into these topics would help us understand the position of the practitioner and their struggles better, and help guide future research.

3.1 Procedure

The findings presented in this paper result from a multi-step data collection process. From July 2014 to October 2014, two online questionnaires were sent out to 2,500 members of the hosting site GitHub[1]. Users were selected based on whether there were recently active on the site, had made their email address public, and whether they had set an organization in their profile, increasing the probability of finding professional developers.

For better understanding and deeper insights, we interviewed 22 professional software developers via Skype calls. These semi-structured interviews

[1] https://github.com

lasted about half an hour. During the interview phase, we constantly compared our insights to data from earlier interviews and textual answers from questionnaires. This iteratively refined our data collection process. Between interview sessions, we adapted the interview guide to reflect what we had learned. Throughout the collection process, we wrote memos to highlight interesting findings and connections between emerging themes.

The Grounded Theory model presented in this paper has two data sources: textual answers obtained from the two online questionnaires, and a set of transcribed interview recordings. Two researchers, working in close cooperation, identified key points in the interviews. Two random interviews were used as a baseline: each researcher identified key points in them independently and their results were compared. Transcripts and textual data and key points were open coded until we identified the core category *'Practitioner's View of Testing Skills on New Hires'*, at which point selective coding followed. Codes were grouped, leading to the emergence of concepts (such as different onboarding strategies) and themes. These were either attributed to subcategories or the core category (for example, a subcategory was *'Coping Strategies'*). These categories are reflected as subchapters of Section 5, where we present our main findings. Lastly, a third questionnaire from our core findings was created for a final validation by a larger population of practitioners. This final questionnaire and its results are discussed in more detail in Section 6.

3.2 Questionnaires Design

In our study, we sent out three online questionnaires, two while constructing our Grounded Theory model and one for validation purposes. The first questionnaire was exploratory in nature and served to give us a first understanding of our population and its concerns. After analyzing the first results, we refined our questions in the second questionnaire to get a better understanding of emerging themes and environmental attributes, such as the *educational origin of the new hire* or the *size of the company*.

The second questionnaires contained two different kinds of questions: questions about the respondent's situation—to give us a better picture of who we were talking to—and open questions about their experience with newcomers. We asked whether they were satisfied with the testing skills of new hires and, if there were any problems, what was perceived as problematic exactly. Between "yes" and "no", the option of textual answers was given. These textual answers were analyzed and counted them towards "yes" and "no" where applicable. Ultimately, 10 ambiguous responses had to be removed.

The respondents' situation was assessed regarding *company size*, *criticality of their product* and the *level of sophistication* of the organization's testing process: whether it was done manually or automatically, systematically or not. For better context, a more in-depth textual description of the testing process was inquired. Also, respondents were asked how closely they worked with new hires.

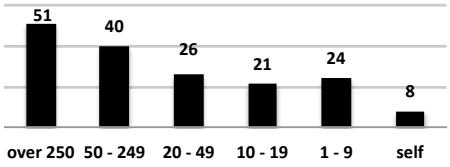


Fig. 1: Number of respondents, split by company sizes (in number of employees).

3.3 Data Collection

In July 2014, we sent out our first questionnaire to 500 active GitHub users who had been active in June 2014, had made their email address public, and had filled in the *organization* field in their public user profile. We received 72 answers by mid-July. Data from respondents who did not come in contact with new hires or who did not develop software professionally and commercially (e.g., pet projects) was removed. 54 responses remained. Of those, 28 respondents agreed to be contacted for interviews and we invited them all, but none enrolled. In July 2014, we sent out our improved second questionnaire to 2,000 active users, excluding the previous 500 users. We received 237 answers by the end of August 2014. After data clean up, 170 responses remained. Of 109 participants who left contact information, 22 enrolled for semi-structured interviews via Skype call.

3.4 Population Description

We were interested in real-world experience with onboarding newcomers. Our target population was software developers currently working at a software company. Our questionnaires were sent to a random set of practitioners who in turn answered on a self-selected basis. This resulted in a diverse population.

Participants were asked to provide some background information of their companies and the testing process they used. The first questionnaire was exploratory and was not analyzed further for the final Grounded Theory model. Here, the population of the second questionnaire (with 170 usable data sets) is described.

Participants were located all over the world, with most of the developers coming from the USA (52 participants, 31%), followed by Brazil (13 / 8%), France (9 / 5%), Germany, and the Netherlands (both 7 / 4%). All in all, developers from 41 countries participated. Practitioners of our study worked at software companies of different sizes. About half of our population worked at a *very large* (over 250 employees) or *large* company (50–249 employees, c.f. Fig. 1). We focused on employed developers but included two unemployed respondents, who shared experiences from former jobs.

We asked our participants to classify their product in terms of criticality. Most of our participants (117 / 69%) worked on products that caused monetary damage if they failed. 39 (23%) participants classified product failure as *non-critical* ('it is not the end of the world'), and 14 practitioners worked on *safety-related software*. Also the respondents' level of sophistication regarding their testing process was inquired. A large number of practitioners (109 / 64%) used *systematic* and *automated* approaches to software testing. All other combinations remained equally at 12% (about 20 respondents for each combination).

Most of our respondents (106 / 62%) hired newcomers that came directly from some kind of educational institution, such as recent university graduates. 64 respondents focused on hiring newcomers from other sources, such as modern development job offer platforms (e.g. odesk.com), headhunters, through engagement in open source development, or through personal connections.

3.5 Fundamental Definitions

The term *testing skills* by itself is ambiguous. In the context of this study, *testing skills* encompasses two notions: *theoretical* knowledge of systematic software testing and *practical* skills in automating systematic software tests.

Theoretical testing knowledge describes a newcomer's general awareness knowledge and how-to knowledge of systematic testing techniques. Being aware of the existence of certain testing techniques and having knowledge about testing-related terms enables newcomers to search for tutorials and testing-related tools if necessary. Awareness knowledge alone is insufficient for a vigorous application of a new testing technique. Newcomers need an understanding of *how* a certain testing technique works, e.g. what the underlying principles of applying this technique are. Usually, it is the goal of educational institutions to provide this kind of awareness and how-to knowledge. For example, at university, students learn about the existence of unit testing and what unit testing is, its benefits and caveats and how to do it.

In order to apply such testing techniques, a newcomer needs an understanding of the general idea of systematic testing. This includes knowledge in designing a test case and being able to devise a correct test case for a certain situation. The newcomer must be able to design a valid test scenario, engineer a correct test input, determine a corresponding test output (according to requirements), and provide a valid test setup and tear down. Test inputs and outputs are not limited to numerical input but can—for example—also take the form of complex (graphical) interactions with the application under test.

When a newcomers enter the working life, another facet of her testing skills steeply increases in importance: practical test automation skills, e.g. the ability to actually act on her testing knowledge and implement it in a real-world situation. Writing test code for a real software project is much harder to do than writing test code in an isolated and prepared educational setting. Implementing test code demands knowledge of the existence of features of the

test framework and knowledge of how to use them correctly. For example, automatically open a browser is a feat that a newcomer must overcome in order to implement an automated GUI test case in web development. Additionally, contextual knowledge for this test case is needed. A test case for bigger commercial projects is not written in isolation, but demands a fitting and executable test setup and tear down. For example, a test may demand a specific database state and the newcomer must figure out how to set it and where to get valid customer data.

In the course of the discussion of this work, the term *social transparency* [41] will be used and a short definition is provided here: Social transparency is the *availability of social meta-data surrounding information exchange* and defines the degree to which a user's actions are visible to other users. This framework helps to adapt the concept of being aware of another collaborator's actions to the characteristics of modern, digital collaborations—such as social networks, global software engineering, and social coding platforms. For example, on online social coding platforms (e.g. `github.com`), users develop software and interact with one another under a high degree of social transparency: Users can follow each other, analyze and take part in discussions, observe what artifacts other users produce (e.g. code snippets, whole commits,...) and adapt solutions devised by other users.

4 Analysis of Questionnaire Data

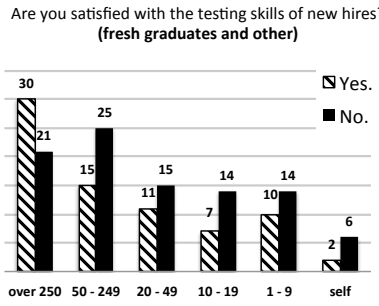
Our second questionnaire inquired: *"Are you satisfied with the testing skills of new hires?"* Here, this numerical data is presented, describing the satisfaction among practitioners from different points of view. Our population is self-selected and our data provides little in terms of statistical significance. Therefore, this data is presented for illustrative purposes only.

Overall, more than half of the practitioners of our study (95 / 55%) are dissatisfied with the testing skills of new hires.

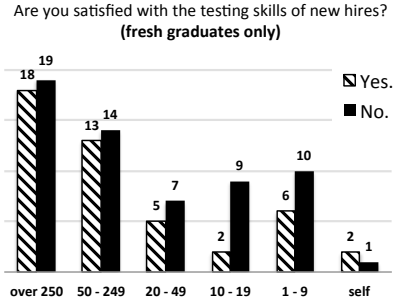
Splitting by *company size* (c.f. Fig. 2a), dissatisfaction surpasses the rate of satisfaction in most cases. From large companies (50–249 employees) to very small (1–9 employees) companies, practitioners are discontent with the testing skills of new hires. Only for respondents from very large companies (more than 250 employees) the majority is satisfied.

Splitting our data by the origin of the new hire (*"Where do your new hires come from, mostly?"*, c.f. Fig. 2c), we see that *very large* (over 250 employees) and *large companies* take in quite a high number of new hires straight from educational institutions (e.g. universities). This is not as obvious for *mid-sized* or *small* companies: The ratio between fresh graduates and new hires from the job market is nearly 1:1. Interestingly, the rate of graduate hiring spikes again with *very small companies* (1–9 employees).

Regarding testing skills of graduates only (c.f. Fig. 2b), dissatisfaction seems prevailing and is most prominent in *mid-sized* (20–49 employees) to



(a) General satisfaction with testing skills of new hires.



(b) Satisfaction with fresh graduates' testing skills.

(c) Educational origin of new hires per company size.

Fig. 2: Split by company sizes: ratio of satisfaction with testing skills, (a) and (b), and origin of new hire (c).

very small companies (1–9 employees). *Very large* and *large* companies seem equally content as discontent with graduates' testing skills.

All in all, practitioners in our study seem to be rather dissatisfied with the testing skills of new hires. Especially mid-sized to very small companies seem to struggle with recent graduates (c.f. Fig. 2b). Here, dissatisfaction rates were equal to or surpassed satisfaction rates. Larger companies seem to fare better than smaller ones.

5 Findings

We wanted to better understand this phenomenon of dissatisfied practitioners and set out to research the question, "What are the testing skills of new hires, as assessed by practitioners?" As per Glaser's idea that "everything is data", we analyzed 22 interview transcripts as well as 170 textual questionnaire responses and used insights from both data sources to develop our Grounded Theory model.

Practitioners' experiences in handling newcomers has led to certain views on their testing skills which in turn spawned different coping strategies (c.f.

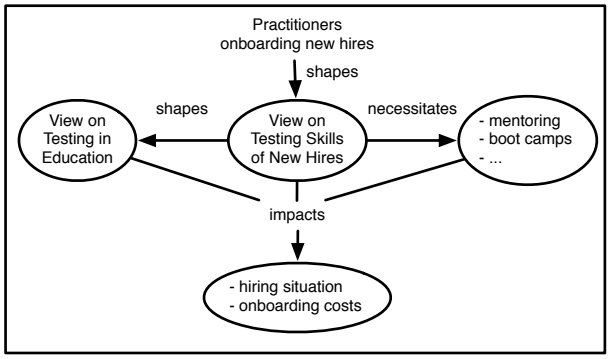


Fig. 3: Overview of the interplay between practitioners' experience with new hires' testing skills and its impacts.

Fig. 3). Ultimately, these phenomena influence the hiring process for both the practitioner and the newcomer.

5.1 Practitioners' Views on Testing Skills

Practitioners find that interest in development generally surpasses testing efforts and testing engagement in new hires. Newcomers have difficulties putting their theoretical testing knowledge to work and implement actual tests. Fundamentally, new hires display *gaps in testing knowledge* and a *lack of hands-on experience* and training. This impression has already solidified in the minds of practitioners, so that it negatively influences the expectations they have regarding the testing skills of new hires when those apply for a position. Ultimately, practitioners *blame the educational system* for this lack of practically relevant testing skills.

5.1.1 General Views and Expectations

New hires applying for a software engineering position do have experience implementing and building software. For example, pre-graduate students sometimes work voluntarily on open source projects or small personal projects. However, testing skills are usually low in comparison and unsystematic manual testing is the prevailing method for quality assurance. Practitioners rarely find good test automation skills in new hires.

> *"They generally have been building for the Web or small personal projects and their concept of testing is 'try it and see if it works."'* — KG

Worse yet, there are practitioners who assume that recent graduates will have no testing skills at all. They are under the strong impression that newcomers

usually have no knowledge about testing techniques and test automation. They have come to expect no testing skills in new hires.

Test knowledge and automation skills are regarded as skills that new hires will only learn 'on the job', not in university. This notion is usually given with discontent and in a bitter manner: These practitioners would prefer university graduates to have more testing skills but are being disappointed again and again.

> *"New hires never have sufficient experience with the test workflows in the development process. [...] These sorts of skills are acquired on the job in industry and not at university."* — TR

5.1.2 Lack of Experience

One reoccurring complaint from practitioners is that new hires have no real-world experience in systematic testing. This is a more global complaint that can be split into two concrete situations: experience in *actually writing tests* and experience with following an *overall test process*.

Regarding *actually writing* test code, practitioners find that new hires seldom have had hands-on experience and assume that they have never implemented tests before.

> *"Automated testing is an entirely new concept to most new hires. High-level test suite design and real-world experience is universally lacking."* — CH

New hires often do not know *what* to test and *how* to do it. They *do* have a general idea of testing, but have problems putting their knowledge into action. Their test code does not quite hit the target and usually under- or over-asserts. Practitioners then deem their test code impractical and inefficient.

Our study participants also noticed that newcomers display weaknesses in fundamental test knowledge, such as *test-related definitions* and their purposes. Their educational knowledge of systematic testing appears to be limited to unit testing and does not cover other, more sophisticated test types. For example, they cannot distinguish between integration tests and system tests. Practitioners need newcomers to break free from such a limited point of view and advance to behavior-driven testing—for example, in order to automatically test complete user interactions.

Regarding new hires' *understanding of test processes*, our participants drew a similar picture. Inexperienced new hires—and especially new graduates—have had no experience with real-world projects and their testing processes. They have no understanding of the role and importance of a well-defined testing process in commercial software development.

> *"They know what testing is but often don't understand how important it is in big projects. They are not used to big scale testing processes."* — AL

From the newcomer's point of view, systematic automatic testing is not *integrated* with the main software development process.

In our previous study [29] we made a similar observation. Systematic testing was regarded as something entirely separate from the process of implementation, and was done—if at all—at the end of development. This kept students from experiencing the benefits of automatic testing during development, such as tests catching regressions or the influence of testing on architecture.

Some practitioners, however, have ambivalent feelings towards the lack of real-world experience of new hires. After all, these are recent graduates and not senior developers. Often, these practitioners pointed out that the new hires they had been working with were fast learners and easy to teach.

> *"University students have not had much experience writing tests for information systems. Limited experience, but they learn quickly."* — FE

5.1.3 Attitudes Towards Testing

Underpinning the lack of hands-on experience of newcomers, practitioners have observed a problematic attitude towards testing in general. Testing is regarded as unimportant additional work and as fundamentally not really needed. Often, newcomers prefer a leisured approach to bug discovery altogether and do not take testing seriously. This touches upon a more fundamental problem: For newcomers, the role of testing as a discipline and as an integral part of software engineering remains unclear. In our previous study [29], we found a similarly skewed view: Students preferred to implement new features and regarded automated tests as technical debt. Testing did not seem as important as implementing new features.

> *"A large majority of new hires [...] do not have any concept of testing as a discipline. It is not so much that they are unaware of a particular testing methodology; it is that they have never conceived of testing as something that would even have methodologies."* — RE

Practitioners hypothesize that the reason for such a dismissive view is that these young developers have not had a chance to recognize the benefits of an automated test suite yet, due to their lack of first-hand experience. For example, they have not yet dealt with things like regression bugs.

Another issue adding to this situation is the perceived reward of test efforts. When new hires *do* write tests, they have the impression that they do not benefit from them. In our previous study [29], we saw something similar. Adding a test suite late—for example, at the end of the development process—left our student developers disappointed by the low failure detection rate. All bugs had already been found by trial & error during development.

> *"They do not think they benefit from testing. [...] they actually write tests, but they don't get any benefit from it. What I called benefit is preventing failures in production, preventing bugs."* — AS

5.1.4 Practitioners' Views on Education

Some of our study participants said they were disappointed by the low level of testing skills in new hires and blame educational institutions. A subset of them had gained the impression that software testing is not taught at universities at all.

> *"We don't study testing and we don't know anything about testing in the university, so junior developers have nothing at all about testing. They don't know there is something in the development process named testing, actually."* — AB

Practitioners see several problems with education in software testing. They feel that it is outdated and not adequate for their industrial needs and today's pragmatic standards.

In line with the lack of hands-on experience, software testing education in university is perceived as too theoretical: Staged examples used in testing lectures do not cover the range of skills needed for testing in the real world and are too simplified. This way, students may develop a general idea about testing, but they cannot apply it. Real world testing may demand a whole new set of technical skills and the use of test libraries and frameworks.

> *"Real human interaction, say ticking on a web page or tapping on a mobile device, [...] connecting to databases, mocking web connections [...]. They're the real sorts of 'juicy bits', which are very difficult and they've had very little exposure to."* — LS

Practitioners feel that testing lectures in university are not given the importance they deserve. They criticize that taking a course in software testing is only optional when studying software engineering. As per their experiences, students can skip those and avoid learning something about testing in their studies altogether.

5.2 Coping Strategies

Different coping strategies emerged that companies employ to overcome this experience gap. The general goal is to help the inexperienced developer learn how to work in the company as effectively and efficiently as possible. Different approaches gradually push the load of engagement from the new hire to the practitioner—with varying advantages and effects.

In the worst cases, *no onboarding* was offered at all and the new hire was left to overcome problems by herself. Other strategies encompassed *teaching the new hire* how to systematically test from scratch. We divided these strategies into two types: *semi-active* and *active strategies*, depending on the level of support that the practitioner had to provide to the newcomer.

Semi-active strategies provide the newcomer with support from a senior developer, but only in an ad-hoc manner: the newcomer has to ask questions

herself and learn pro-actively (e.g. in mentoring). Active strategies require the practitioner to actively teach the new hire. Here, the effort lies on the practitioner's side. As the level of support on the practitioner's side grows, the learning effect may grow as well. However, so would the effort and induced costs.

Besides coping strategies to remedy the lack in testing skills of new hires, practitioners had *worked around it altogether*: Inexperienced new hires were not required to be able to automatically test their software but were specifically expected to do only manual testing during their onboarding phase. This would give them time to learn about systematic testing and allow them to advance to automatic testing later. This strategy, however, demands that the company's test process and quality assurance system can handle and accept *only manually tested* software written by newcomers. For example, the source code handed in by a recent hire may need to be tested by a senior developer afterwards.

5.2.1 Environmental Properties

Practitioners find a test-intensive workplace to be effective in diffusing testing practices: New hires adopt testing faster if they are put into teams that already practice automated testing. In such an environment, new hires can observe and study how to test, provided team dynamics and commit mechanisms permit them. However, left to their own devices, new hires with low testing skills do not adopt testing well.

A similar effect emerged in our study on testing behavior on GitHub [28]: the site has a high degree of social transparency and makes other users' actions visible, including how they write tests. Seeing that other users engaged in testing efforts, being able to analyze their test code, and use it as a basis for their own tests often had contributors engage in test efforts themselves. We believe that making the test culture in a team visible and actionable lowers the barriers for inexperienced new hires to adapt it.

> *"They are quick to adapt into doing auto testing if they work in teams that does so already. But if they are put alone in a new project it does not come naturally to do any testing at all."* — SW

Another important aspect is the documentation of the testing process. Short and textual how-tos describing tools and how to operate them allow newcomers to orient themselves and help in answering basic questions. This in turn can lower communication load during the onboarding phase. However, good documentation is worthless if the new hire has no access to it or cannot find it by herself. Especially unguided onboarding is dependent on strategic placement of guidelines, documentation, and communication thereof. Another challenge is to organize these guides in an accessible manner. Here, practitioners preferred short and pragmatic texts. If the barrier for using documentation is too high, a newcomer may not benefit from it.

> *"There were documents on how to test, how to write tests that fit into our automated framework. [...] The problem with the wiki was that it wasn't well organized and there was an awful lot of pages."* — KG

5.2.2 Semi-Active Onboarding Strategies

During onboarding, specific questions arise that documents cannot answer. In this case, one semi-active strategy that our study participants mentioned was to assign a senior developer to the new hire as a mentor. The *mentor's* task is to get the new hire up to speed, explain company-specific processes, and help the new hire with technical questions. The new hire is expected to ask questions when something is unclear, which can be a problem: Being a mentor is an additional role for a senior team member and as such, this can impact this team member's productivity. New hires are aware that they are occupying the senior's time and they do not want to impose too much. This makes asking questions for a new hire difficult and stressful. They do not want to keep the senior developer from completing their work.

> *"You need to understand everything very, very fast. You will almost not get a chance to ask this really deeply. So you need to prepare your questions before, just not to take too much time and ask everything you need at one talk."* — IO

5.2.3 Active Onboarding Strategies

Strategies for onboarding that involve active teaching efforts on the company side ranged from seminars to carefully curated sample projects. Most importantly, every project endeavor with the new hire used the company's development process and was designed to make the newcomer become acquainted with it.

One teaching technique used was the *bootcamp*. New hires are categorized by proficiency and strategically grouped to facilitate knowledge transfer between participants. Then, they are sent off to a quick hands-on crash course. They implement a small but valuable project for the company including testing efforts. Often, these courses are located in-house and newcomers are supervised by senior developers. They provide a good starting experience: Newcomers create a viable product and feel proud of their accomplishments.

Hosting and managing such a specifically curated course in programming, testing, and the company's processes was said to be effective and to help the new hires overcome testing skill gaps. However, the associated efforts and costs were said to be relatively high. Companies of small size may not be able to conduct such an event for onboarding new hires.

> *"A lot of the new hires are proud of that and they like to present what they have build and more important, more of them present what they have learned."* — RS

If no bootcamp is available, small and manageable projects are used as a playground for improving the newcomer's skills. These projects are non-critical and sometimes not related to the company's product at all. Practitioners are aware that newcomers are not able to perform on commercial products straightaway. They do not want to overwhelm the newcomer and try to provide a *low pressure but real* environment where newcomers can train their programming and testing skills.

> *"[being thrown in] is a little bit of dancing for developers as well, especially if you come straight out of university ... to be put on client work and to be expected to deliver a level which you're probably not able to deliver at."* — LS

Another approach our participants reported is putting the newcomer on a new project with a fresh code base, with a senior developer taking the lead and reviewing their work. The newcomer is not taking full responsibility and can safely experience the company's development process.

When it comes to diffusion of culture and team knowledge, pair programming seemed popular among practitioners. Practitioners said they were able to show how things work in detail, while at the same time seeing how the new hire approaches problems. Dubbed *cross pollination*, senior developers not up-to-date with recent technologies and practices in systematic testing were paired with a newcomer to refresh both their testing skills.

5.3 Impact of Low Testing Skills

The gap between what is provided by university education and industry's needs has different impacts on the onboarding phase. First, teaching practical testing is *time-consuming and overall costly*. Onboarding an inexperienced developer can negatively influence a team's productivity. Second, a perceived low level of practical testing skills in graduates has given practitioners a *negative impression* of testing education. Third, not all companies decide or can afford to take on the effort of onboarding an inexperienced developer. In some cases, the hiring process had changed and the *hiring bar had been raised*, excluding inexperienced developers altogether.

5.3.1 Costs of Teaching

Practitioners care for code quality but do not find appropriate testing skills in new hires. However, specific lecture courses, bootcamps, or mentorships are costly in terms of both time and money. Generally, these endeavors are more effective if the new hire is capable and willing to learn. In line with what Singer et al. [37] have found about how recruiters assess software developers, practitioners often look for 'quick learners' and try to filter for 'quick learning' abilities when interviewing. Yet, teaching testing from scratch remains effortful, even with quick learners. Also, not every capable graduate computer science student may pass as such a quick learner instantly.

> *"Fresh graduates almost never have any testing skill or experience. We make a point of recruiting quick learners, so they pick up automated testing fast, but it takes up quite a bit of energy."* — SH

Mentoring is costly: often a senior developer is appointed as a mentor. This additional role can impact productivity, which can especially hurt small teams.

> *"We have three guys and when we hire another one, we stop one of the three guys to basically teach this new hire from the scratch. So in a team that have three guys we have one off for a bunch of time. It will suddenly impact the speed of all."* — FR

Such real-world onboarding strategy could point to a deeper organizational problem masked as an onboarding issue: for smaller development teams, it could be more beneficial to evenly distribute the effort of mentoring to all team members. This, however, is an insight that these companies need to be made aware off, before they can take action.

5.3.2 Raising the Hiring Bar

Onboarding efforts and associated costs influence hiring strategies. For some practitioners, it is no longer sensible to hire inexperienced developers. They increase their hiring bar in the hope of reducing training efforts and costs. Questions about testing are included in interviews and ad-hoc testing exercises are demanded. However, finding candidates with both acceptable engineering skills and testing skills proves difficult.

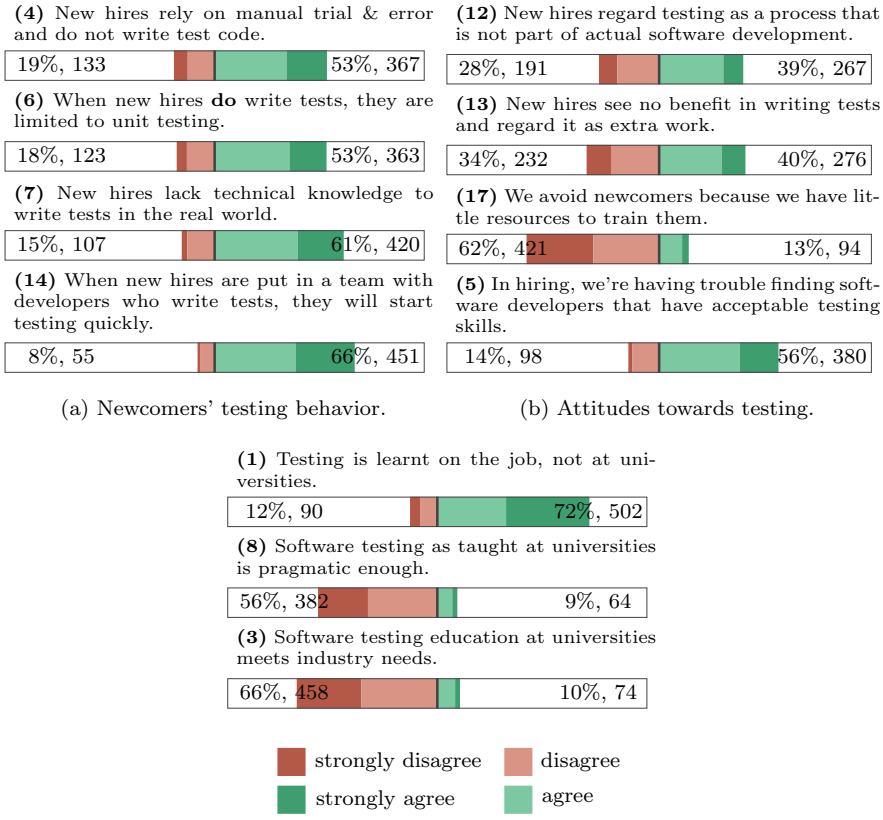
> *" there just didn't seem to be so many candidates who had both experience in engineering and had experience in any sort of meaningful testing."* — JC

The opposite approach is to exclude recent graduates altogether. Several of our participants said they were content with paying more for more experienced developers rather than having to onboard newcomers. This, however, makes it hard for recently graduated university students when applying for a position at one of these companies. They have little practical experience when finishing their studies, their potential employer however demands such experience. Larger companies with more resources can offer training courses and bootcamps, but smaller companies often cannot, instead raising the hiring bar.

> *" after you graduate you just cannot find a work because at work no one wants to teach you from the beginning and what you are learning at university [...] if you don't know how to code simple things, then who will hire you?"* — IO

6 Validating Core Findings

In order to validate our Grounded Theory model, a final third questionnaire was designed. We translated core findings into 14 statements that we asked



(a) Newcomers' testing behavior.

(b) Attitudes towards testing.

(c) Practitioners on testing education.

Fig. 4: Practitioners' views on testing skills of newcomers.

questionnaire respondents to rate on a five-item Likert-type scale ranging from "strongly disagree" to "strongly agree." In July 2015, the questionnaire was sent to 10,000 GitHub users who had set an organization in their profile and received 941 answers. After sorting out hobbyists, practitioners not in contact with new hires, and only manually testing participants, 698 usable responses remained.

As with our second questionnaire, participants were located all over the world, with most of the developers again from the USA (217 / 31%), followed by Germany (43 / 6%), Brazil (37 / 5%), UK (36 / 5%), and India (30 / 4%). All in all, developers from 75 countries participated.

Our results are shown in Fig. 4 and Fig. 5. Agreement and disagreement are represented by shaded bars, percentages, and absolute numbers. Neutral answers are not shown explicitly, but can be derived from the white space in a chart. This representation was chosen to emphasize agreement and dis-

agreement. Statements 2, 10, and 16 are not included[2]. Statement 2 is left out because statement 5 (c.f. Fig. 4b) is a more testing-focused version. Statements 10 and 16 were control questions. The numbering of the statements indicates the ordering of the statements in the questionnaire. In this section, they are grouped by topic for a more cohesive presentation.

Participants of our final questionnaire were largely dissatisfied with the testing skills of new hires (*"Are you satisfied with the testing skills of new hires?"*: 359 / 51% dissatisfied vs. 293 / 42% satisfied, 46 no answers).

Working with newcomers. Practitioners' experiences with writing tests with newcomers largely align with our findings (c.f. Fig. 4a, statements 4, 6, 7). When it comes to *actually* writing test code, new hires appear to be limited to unit testing to a degree, lacking knowledge and experience to write tests in a real world situation.

Attitude towards testing. Regarding new hires' higher level understanding of industry test processes, practitioners' opinions diverged a little from our findings (c.f. Fig. 4b, statements 12, 13). There is no decisive majority to either agreement or disagreement regarding our statements 12 and 13, although a very slight skew towards agreement can be observed. Practitioners appear to encounter both new hires *with* and *without* a proper understanding of high-level testing processes in a nearly equal manner. In our interviews, interviewees complained about a lack of general understanding of industry testing processes and their importance. Additionally, we found the same phenomena in previous research when interviewing students about their test behavior [29]. For more decisive conclusions, further research is needed.

Strategies. Our participants agree that being put in a team that already practices systematic testing facilitates the adoption of testing (c.f. Fig. 4a, statement 14).

Impact. The lack of testing skills in newcomers has impacted the hiring situation. Most participants of our questionnaire struggle to find newcomers that meet their testing needs (c.f. Fig. 4b, statement 5).

Rectifying a lack of testing skills is costly as mentoring or training endeavors cost either productivity, time, or money. Our previous questionnaire results (c.f. Section 4) and interview data suggested that the extent of this struggle differs depending on company size. To this end, we designed this final questionnaire to validate our hypothesis: *Smaller companies struggle more with onboarding new hires without testing skills than bigger companies.* Participants were asked about the size of their company in number of employees and assigned size ranges into five different categories: 'working alone (self)', '1-9', '10-19', '20-99','100 or more'. We included two opposite statements of our hypothesis (c.f. Fig 5, statements 9, 15) and analyzed results with respect to the participants' company sizes.

[2] The statements not depicted are:
(2) In hiring, we're having trouble finding software developers that meet our needs.
(10) Testing is an integral part of software engineering.
(16) Testing is something that is separate from actual software development.

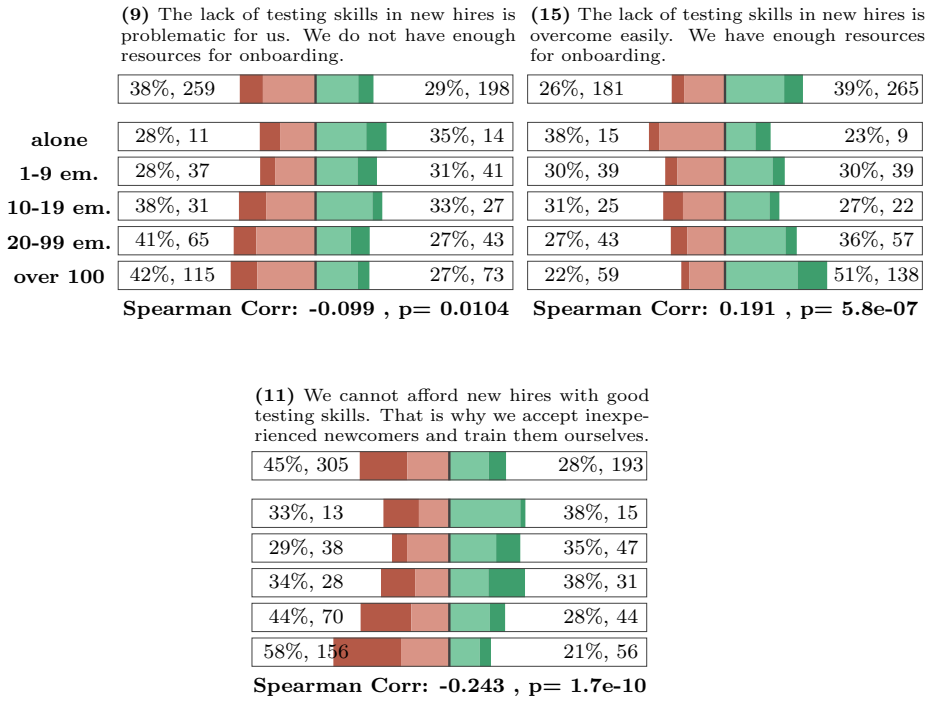


Fig. 5: Practitioners' struggle with the lack of testing skills, split by company sizes in number of employees.

Did participants from smaller companies answer differently than those from larger companies? Fig. 5 shows our overall population's answer on top and split by company size below. We did find a weak correlation between our participants' answers and their company size. Participants from the two smallest company size categories appear to struggle with the lack of testing skills. Participants of the next company size tier appear to struggle less with resources for onboarding. Larger companies (20-99 and over 100 employees) seem to fare well and mostly say they overcome the lack of testing skills easily.

The Spearman Rho Rank Correlations is given below each diagram and indicates a statistically significant correlation between responses and company size ($p < 0.05$). In this population, we may have been able to validate our hypothesis: smaller companies perceive the lack of testing skills and its demand for resources differently. However, we expected our results to show a greater effect. While we assumed larger companies to fare better, we expected a greater and more distinct need for onboarding resources from smaller companies as suggested in our interview and previous questionnaire data.

Regardless of company size, our population mostly disagreed on rejecting newcomers to avoid training expenses (c.f. Fig. 4b, statement 17). Why is it that even though practitioners appear dissatisfied with newcomers' testing skills, they still hire and train them instead of focussing on better trained alternatives? Practitioners have trouble finding new hires with acceptable testing skills (c.f. Fig. 4b, statement 5)—and demanding better testing skills makes hiring more expensive.

Smaller companies more often opt to take in new hires and train them, simply for a lack of budget for better trained alternatives (c.f. Fig. 5 (statement 11)). Meanwhile bigger companies appear free of this notion which is in line with our previous finding: the lack of resources for onboarding and training is not a big obstacle for larger companies, while practitioners from smaller companies perceive it as challenging.

View on Testing Education. Lastly, practitioners who were in contact with recent university or college graduates strongly felt that testing is only learnt on the job (c.f. Fig. 4c, statement 1). They deemed current testing education not pragmatic enough and said that their needs are not met (c.f. Fig. 4c, statements 8 and 3).

7 Discussion

In this section, possible causes for the low testing skills of newcomers are discussed and improvements for both academia and industry are proposed.

There is a whole spectrum of educational settings and industrial onboarding environments. While there are universities and software companies that cannot relate to the findings of this study, there are those parts of industry and academia that are directly affected by it. For example, before a company grows and is able to provide a well-organized onboarding process, it will have to start small and accomplish onboarding with less man-power or budget. This discussion may be of help to the latter groups.

7.1 Analysis: What Factors Lead to This Situation?

There are both attributes of the educational setting as well as attributes of the activity of systematic testing itself that hinder its vigorous adoption and implementation by newcomers.

In the educational setting, time and organizational constraints make conveying the real world importance of testing difficult. In the short time that they have available, lecturers focus on conveying the fundamental theory of testing first and the technicalities of it second. From their point of view, technical trends change too quickly while core principles stay the same and are more important. However, systematic testing is not a main focus in nowadays' software engineering curriculum, but only a small part. At many universities, courses on software testing are not even mandatory.

Students are likely to perceive the idea of systematic testing as *unattractive* when being first introduced to it in class. Rogers' Theory of Diffusion of Innovations [34] can be used to explain this claim. It explains how and why tools, practices, ideas or technologies are adopted or rejected by individuals or groups. Rogers documents properties of such *innovations* that support their adoption:

- *Relative Advantage:* If an innovation has clear advantages over known alternatives, it is more likely to be adopted.
- *Compatibility:* Being in line with previous experiences and needs of an individual increases its chances of being adopted.
- *Complexity:* If an innovation is perceived as too complex, it is less likely to be adopted.
- *Trialability:* Being able to try the innovation before deciding to adopt it increases its chances of being adopted.
- *Observability:* Being able to observe other adopters using the innovation successfully increases its chances of adoption.

In the educational setting, there is not such a strong focus on code quality and the students' livelihood is not dependent on it. A student's income does not depend on the software the student writes in class; time constraints are not as strict, and so on. The student's situation does not demand or raise any real need for systematic testing. They have not worked in a real software development project before and are not be able to relate to the importance of systematic testing for it. This decreases the perceived relative advantage and compatibility of the idea of systematic testing.

In the lecture context, students perceive the complexity of systematic testing as low—even though the engineering discipline of systematic testing is non-trivial in the real world. The high complexity of systematic testing is masked by a lack of real project context: practical exercises accompanying lectures are usually kept simple in order to facilitate a better understanding. Often, exercises demonstrate techniques and topics in isolation without any dependencies to other software modules. This reduces the barrier of having to overcome technicalities which are present in the real world. The exercise context is often prepared for students to interact in it easily. Thusly, trialability is perceived as high.

Observability of systematic testing may be perceived as high — for example, if students talk about the current exercise. However, this observability is more related to solving of the exercise and not related to demonstrating success with practicing systematic testing.

Lastly, systematic testing is a preventive effort. It has clear-cut upfront costs with benefits which may or may not be realized later on. Such preventive innovations are generally more difficult to reach widespread adoption.

The graduate's situation changes when she enters the onboarding phase. She is confronted with a real-world software development project — including a real project context. She has a personal interest in the success of this project and her livelihood is dependent on the quality of code that she produces. In this

setting, she understands the importance of testing better and feels a need for systematic quality assurance. This increases the perceived relative advantage and compatibility of the practice of testing.

However, this introduction of project context also steeply increases the perceived complexity: Testing in a real project demands handling and overcoming of technicalities. Contextual knowledge is needed, for example, where to find valid test data and how to connect to the test server in a particular project. If certain techniques have not been covered in education, the newcomer needs additional materials to learn from scratch. With this rise in complexity, trialability of systematic testing decreases. Setting up a runnable test project is not as trivial as in the curated exercises of the educational situation.

Observability of systematic testing in the new development team is difficult to assess. It depends on the team structures and may increase: does the team talk about how they test often? Do developers show each other successes related to testing? Or does every developer just writes tests on her own? Regardless of its level, observability is now related to actually practicing systematic testing and not to solving exercises which is more suited to demonstrate its usefulness to newcomers.

The combination of a perceived high complexity, little trialability, and a fundamental lack of how-to-knowledge can be a hard blocker for a fast adoption of systematic testing in the onboarding situation.

7.2 Industry: Improving the Onboarding Situation

We propose to strategically improve factors that keep newcomers from engaging in systematic testing: raising observability of systematic testing and lowering the perceived complexity, while raising its trialability.

Raising Observability: The first goal can be achieved by communicating the testing culture efficiently to the newcomer. A better understanding of how and why other developers practice systematic testing may even raise its relative advantage and compatibility further. We propose to communicate the team's testing culture by strategically employing traits of social transparency [42] into the newcomer's work environment. One of our core findings is that newcomers seem to be able to pick up systematic testing quickly—when they are put into teams that practice it already (c.f. Fig. 4a, statement 14). We suspect that there is some degree of social transparency given in these teams and that newcomers have some way of understanding how and why senior developers engage in writing tests.

During onboarding at a company with a well-established testing process, communication of the testing culture can happen naturally: By taking part in and by being exposed to testing-focused practices, such as systematic test code reviews or adhering to process-required test coverage criteria, the newcomer can observe certain aspects of the team's testing culture. We assume that the level of maturity of the test process could influence the adoption of testing

culture by newcomers. In matured development and testing processes—such as proposed in [16]—systematic testing is given more exposure and visibility, increasing opportunities for the newcomer to engage in testing. However, we were unable to find research on this effect in literature. Additionally, not every onboarding setting features such practices—even though the team is engaging in systematic testing. While other common practices, such as the ability to access and read other developers' test code stored in team's versioning system, also support communication of testing culture, this effect is a by-product and not guaranteed: a newcomer may not turn to the versioning system on her own to understand how senior developers test—even though valuable information is stored there. By explicitly raising the observability of systematic testing efforts in a team, this communication of testing culture is made systematic, predictable and manageable.

Researching the effects of a high degree of social transparency on the testing behavior of developers on a social coding site [28], we found that being able to see other users' testing effort, to follow their discussions and their rationale improved testing efforts of developers. From these insights, we extracted the concept of testing signals [30]: Small signals placed in a newcomer's IDE that graphically communicate the testing effort of other team members. Six testing signals were implemented and their effect on students nearing the end of their Bachelor studies was evaluated with encouraging results: Seeing other team members' testing effort made them aware of the current testing culture and made them increase their own testing effort.

Lowering Complexity, Raising Trialability: The second goal can be achieved by strategically providing the newcomer with technical knowledge on how to practice testing in this onboarding situation. Coming from university, they lack technical experience and explicit how-to-knowledge. We propose to automatically recommend the newcomer with contextually relevant and executable examples of test code from the current project. This way, newcomers are enabled to study senior developers' test code and to understand how to write tests in this environment. On a social coding site, open source contributors heavily rely on existing test code to write their own [28]. Being able to access existing test code greatly facilitated the writing of test code. In the onboarding context, it is important to recommend only test code examples that are easily runnable in the newcomer's environment. Any transfer effort increases the barrier of adoption: Studying the testing behavior of students, we found that students have difficulties in finding suitable testing tutorials online [29]. Often times, they were not able to apply these examples and gave up. In this sense, the trialability of these code examples needs to be high.

We have implemented a test code recommendation system and evaluated it with positive results [31]: After analyzing the changes that the newcomer made to the code base, the recommender system proposes a piece of test code from the test suite to the newcomer to serve as a suitable working basis for new test code. Students liked being presented with test code, browsed test code for a better technical understanding and were able to produce new test code much

faster. Especially shy newcomers—not wanting to hassle their mentor—can benefit from being able to procure a test code example automatically.

7.3 Academia: Dedicated and Practical Testing Courses

As a more general remark, we propose to make the topic of the onboarding situation a part of the software engineering curriculum. As seen in our qualitative data, newcomers are overwhelmed by the new tasks demanded from them. Such insights as newcomers being too intimidated to ask their mentors for help or their struggle with outdated wikis could be made a subject of discussion in software engineering courses. Being prepared for this situation explicitly can help to lessen the impact of the new environment.

Still, academia can take more far-reaching actions to bridge the gap between graduates skills and industry's expectations. While industry puts emphasis on technical implementations—practitioners in our study are disappointed by the technical know-how of newcomers—we do not advocate to solely focus on industry's wishes. Technical implementations in industry change quickly and are subject to trends and fast innovation cycles. Letting that guide the education of our engineers can bring by unforeseen consequences[3]. The divide between principle and actual implementation is important. But, seeing that industry moves fast and our community *wants* software engineers to efficiently work in this environment, this divide needs constant attention.

We propose a two-phased model to approach the problem of elevating graduates testing skills to industry's needs while maintaining academic independence: a *dedicated* and *practical* course on systematic software testing. This course model targets two goals:

- students understand the underlying principles of systematic testing first.
- students are confronted with writing tests in a real-world environment early on and learn how to overcome real-world testing problems.

Part 1: A Dedicated Course on Testing. Software testing is part of software engineering courses at universities, but courses focusing *solely* on testing activities remain rare. There is a distinct need for a *dedicated* course on systematic software testing in today's software engineering curriculum: For students it is difficult to learn and apply the practice of systematic testing *simultaneously* to other software engineering activities and often systematic testing is the first activity to be left out [29]: During a 4-month development project, students struggled with practicing systematic testing as they had to concentrate on learning how to collaborate in a software development project. They were busy learning team-based development, gathering and documenting requirements, designing, and implementing their software. This left them no time to learn

[3] For example, technologies like Adobe Flash go in and out of fashion, but one would not want a whole generation of software engineers being trained solely in how to work with Adobe Flash.

testing practices at the same time. Just as they had been taught programming before, these findings suggest that there is a need for a purely testing-oriented course that lets students focus on this part of software development.

Similar to exercises accompanying engineering lectures nowadays, this dedicated course targets to convey the principles of systematic testing. Testing techniques can be exercised in small groups of students, supervised by a lecturer. At this stage, it is important to strip the learning experience from distracting technicalities or implementation details of systematic testing. Lecturers can base such a dedicated testing course on existing test-related exercises and expand from there.

Just as a high degree of social transparency can help to communicate testing culture [28], it can also help students learn as a group: students can easily collaborate with other groups, learn from them and share and trace solutions or ideas. This could be achieved with social coding platforms that provide a high degree of social transparency, such as `github.com`. Github is used in educational environment with promising results already [44], for example in engineering classes on software architecture [10].

Part 2: A Practical Course on Testing. As the second part of this testing course, we propose to let students practice systematic testing in a real-world setting, putting emphasis on two aspects:

- students are pushed to apply the theoretical testing knowledge that they have gained in the first part of the testing course, and
- students work on a real software project (provided by industry) and are confronted with testing-related technicalities early on.

This part of the testing course accompanies the development of an industrial project and puts students in the role of dedicated testers. To lower barriers and to facilitate communication and diffusion about testing techniques, students are paired and implement tests in a pair programming manner (dubbed *pair testing*). The lecturer of this course acts as a coach for testing technicalities and guides students through different testing phases (e.g. considering testability at design time, actual test implementation before or at implementation time,...). Students experience the project's development through the viewpoint of an actual software tester.

Working on a real-world project positively influences students' view on systematic testing and improves their general attitude towards it. Students value the feeling of accomplishment when finding defects in an actual project, but do not feel this in lecture exercises [29]. Lecture exercises—such as used in the first part of this testing course—are isolated, focusing only on the principles of testing techniques. They lack a project to give these techniques context. Solving such testing exercises does not communicate a high relative advantage, however, practicing systematic testing in the context of an actual project does—an actual defect is found and subsequently removed.

Introducing a real-world project context must be done with care and in a controlled manner. Overloading the student with technicalities can quickly distract from the actual testing task. Distracting technical issues not pertaining

to the testing task at hand should be excluded. This differentiates this practical course from an internship or an educational software development project. For example, going through a requirements and design phase or maintaining a mysql database before starting to test is not expedient for practicing systematic testing. However, learning how to setup and use features of a test framework is.

The target of this course is to help students build up a technical knowledge of *actually* practicing systematic testing while maintaining a manageable workload for the student. This technical knowledge helps her in overcoming similar technical barriers when onboarding. While, it is not guaranteed that she will encounter just the same testing framework, her hands-on experience should prove beneficial. This will help her to master this new situation more quickly and reduce any feeling of being overwhelmed.

In current software engineering curricula, there are already efforts for providing students with hands-on experience in systematic testing. Software testing courses focused on practicing testing in open source projects [10], classic student software projects [24], or project-based collaboration with industry partners [5] are some examples of these efforts. It is not clear whether the majority of educational institutions is able to offer these kinds of industrial collaborations. Regarding the latter two examples, there is no particular focus on systematic testing which may result in students being overwhelmed with other engineering tasks [29].

7.4 Collaboration between Academia and Industry: Common Body of Model Testing Projects

As a premise for the proposed dedicated and practical testing course, academia is in need of a constant flow of model commercial projects. In order to give students a fairly current impression of testing in industry, the subject project for the testing course needs to be updated regularly.

These projects must fulfill certain requirements to serve as educational material. Their implementation of testing techniques should not be opposite to what is taught in class, the techniques used in the project should not be outdated, and so on[4]. Generally, a manageable (in size, ...) and presentable project is needed that represents common industrial practice.

This re-occurring need for current model projects could spawn a collaboration between academia and industry: Software development companies could submit presentable real-world projects as model projects and academia could establish a database of projects with "good testing practices, as applied in industry". Software companies in the vicinity of the university could participate and have their projects dissected by students—giving them a spotlight to show their high-quality standards and attract newcomers[5].

[4] The compilation of a comprehensive list of quality attributes is left to future work.

[5] Sometimes, companies make their products available as open source in order to show off their high-quality standards and attract capable contributors [28].

Sharing these "best testing practices of commercial software projects" online globally can potentially create a *common body of model testing projects* and diffuse best practices in the software engineering community. Sharing onboarding practices and testing-guidelines between companies is not an uncommon practice[6]. Such a common body of model testing projects would make the discovery of advantageous testing practices more systematic and easier. While academic conferences report on industrial experiences and produce proceedings, attending such conferences is of little interest to smaller companies. Entry fees are relatively high and they may not be able to afford the downtime of a team member. Access to and participation in this common body of model testing projects could help these companies in improving their quality assurance efforts and in attracting newcomers.

This collaboration would offer both academia and industry a low-barrier opportunity to diffuse living software engineering practice into the curriculum while maintaining a focus on academia principles. Other forms of collaborations with industry, such as internships, are harder to come by: Software companies are busy with day-to-day business and a collaboration with educational institutions may not seem lucrative—the industrial side is usually paying for the collaboration. Collaborations between university and industry partners often pursue a certain research question but do not focus on day-to-day business. The software company may not trust a student to work on a commercial product. While regular and long-time collaboration with software companies are most effective in conveying real-world testing experience to students, they represent a bigger commitment and require a certain company size. Smaller companies—while being most affected by the lack of testing skills—may not be able to provide this.

7.5 Impact

Our study is located at the intersection between educational institutions, researchers, and practitioners. For industry, it delivers a comprehensive view on the current onboarding situation that newcomers and practitioners face. It touched upon topics that have been presented in prior studies or that are seen as 'well-known' facts. For example, the use of a wiki where information on company processes and tools is made available is a well-known industry practice. However, as this data was found in using the Grounded Theory approach, these insights were presented nonetheless, striving for an inclusive presentation. While some companies have little problems in onboarding and see only little value in the insights repeated here, others may find them more insightful. A comprehensive view on current testing-related onboarding issues gives latter companies a chance to recognize problem areas that they were not able to identify before. While it is well-known that update cycle of a wiki

[6] The company Airbnb shared their guidelines for newcomers about using javascript and how to test it, `https://github.com/airbnb/javascript#testing`.

can become problematic, it is not guaranteed that affected companies associated this issue with onboarding and testing. Other companies may associate issues with onboarding that are more related to organizational problems: for example, the small team of only three developers (interview participant *FR* on page 17) had to take one person off the team for mentoring and suffered a loss in productivity—one could argue that this is an organizational problem and mentoring should be evenly distributed among the whole team. Making these insights public in a comprehensive way can be beneficial for companies that struggle with similar problems.

While onboarding was perceived as "costly" in our study, we did not talk about financial numbers for the costs induced by a low level of testing skills and cannot make any statement in that direction. As our population is quite diverse, the numbers could have fluctuated greatly. It is not clear whether or not our participants were in the best position to answer financial questions reliably. However, in our interviews, it became clear that participants perceived the cost of managing these problems during onboarding as 'too high' and that participants were discontent with the current situation. This presents a good starting point for further research into this area.

Educational institutions can use our findings to expand lecture programs and put more emphasis on practical training. We have sketched a new testing-focused course model to help raising the testing skills of graduates. Generally, graduating at a university should not implicate little hands-on experience with testing—even if there is no real-world industry project at hand. As an alternative to industry projects, open source projects provide easy access to practical experiences. Particularly GitHub has seen a rise in educational applications [44]. Herein lies a chance for universities to promote open source engagement in students, integrate open source development into lectures [10], and facilitate early contact with real-world projects.

For researchers, our findings give opportunity to focus on mitigating the current "shock of practice" for newcomers further. We presented nowadays' onboarding strategies whose advantages and disadvantages can inspire and guide the creation of new approaches. We have analyzed the onboarding situation in terms of observability of testing efforts, complexity and trialability and sketched a configuration that—according to Rogers' Theory of Diffusion of Innovations—will have the best effect on newcomers. Our findings about environmental influences present a chance for researchers to implement concrete instances of this configuration: Newcomers put into an environment that practices systematic software testing seem to pick it up more quickly (c.f. Fig. 4a, statement 14). A strategic use of social transparency could help support the adoption of testing practices. In another study, we have seen promising results in this regard [30]. Students who were shown the testing efforts of their team members in personalized diagrams in an altered Eclipse-IDE engaged in test writing more.

7.6 The Special Situation of Onboarding

In our study, practitioners had to teach newcomers how to test *from scratch* which is problematic. Onboarding is already a stressful situation in which the new hire has to adapt to the company's development culture and processes. Adding teaching of fundamental engineering practices impacts the onboarding situation negatively: both the teaching load on the practitioner as well as the learning load on the new hire are higher. This can be expensive and frustrate practitioners. Also, smaller sized companies may not be able to afford this.

The actual onboarding phase may seem relatively short and insignificant compared to a software engineer's career. However, if gaps in testing skills are not taken care of early on, they can negatively influence the newcomers' progress in becoming a high-quality engineer.

8 Limitations and Threats

Our study is a first exploratory investigation into the views of practitioners. As a result, an approach based on Grounded Theory was chosen. While saturation in our interviews and questionnaire responses was achieved, it is likely that we did not uncover all possible perspectives. Our population was chosen semi-randomly and the final interview participants and questionnaire respondents were all self-selected volunteers.

As with most Grounded Theory studies, our results are not generalizable and would need to be verified in other populations. Our study provides a view of testing skills as seen by a certain self-selected population. We cannot judge the strength of any of the presented phenomena or effects. The results of our third and final questionnaire can only serve to heighten our confidence in our findings but cannot provide absolute certainty.

Location. Participants of our study are located all over the world. This presents the threat of local and cultural differences. Among a lot of other things, educational institutions and study programs, work values, or industry requirements may differ from country to country. Although clusters can be observed—for example, the majority of respondents coming from the USA—the threat of local influences remains. In this regard, results of this study can only be seen as a first overview. Further research is needed to understand needs of practitioners with a focus on their respective local industry.

Focus on Dissatisfied Practitioners. Our study focuses on practitioners' problems and concerns with new hires. Opportunities and advantages that give practitioners a positive opinion about new hires are rarely reflected. This is due to field of interest that our study is based on. Our numerical analysis (c.f. Section 4) reflects both positive and negative attitudes towards new hires' testing skills alike. As the negative trend dominated our results, we decided to dig deeper into the concerns and struggles of practitioners. Our resulting Grounded Theory model is thus focused on the negative impression. Further research is needed to analyze the positive aspects of new hires' testing skills.

Varying Bucket Sizes. The results of our third questionnaire were split by different company sizes (c.f. Fig. 5). Due to the voluntary nature of this questionnaire, we had no control over the number of participants for each size category, resulting in possibly over- or underrepresented categories. This, in turn, could heighten the influence of outliers on our results.

9 Conclusions

Our Grounded Theory study investigated what practitioners think of the testing skills of young newcomers. We interviewed 22 professional software practitioners and received 170 answers from online questionnaires. From our qualitative data, we studied the practitioners' experiences in more detail and present a Grounded Theory model. Our core findings were validated in a final questionnaire with 698 practitioners.

According to the data we collected, many practitioners are dissatisfied with the testing skills of newcomers. Smaller companies struggle with onboarding newcomers without testing skills, while larger companies are able to embrace them. We suspect larger companies to have more and better resources for overcoming the testing skill gap than smaller companies, and found a weak correlation between perceived lack of onboarding resources and company size.

Problems that practitioners have with newcomers range from gaps in knowledge to negative attitudes towards automatic testing. New hires have a hard time putting their theoretical testing knowledge into practice, struggling to overcome the technical barriers of writing tests in a real-world environment.

Different strategies were found that practitioners use to cope with these challenges. The most common teaching strategy was mentoring, representing a middle path between bootcamps and doing nothing. Practitioners reported that their companies opted to teach newcomers how to test from scratch.

Our findings can help inspire new strategies for onboarding inexperienced developers, could raise awareness of industry's perception of university education, and finally provide a starting point for future research that could investigate some of the raised issues in more detail.

References

1. Begel A, Simon B (2008) Novice software developers, all over again. In: Proc. of the Fourth Int. Workshop on Computing Education Research, ACM, pp 3–14
2. Begel A, Simon B (2008) Struggles of new college graduates in their first software development job. In: ACM SIGCSE Bulletin, ACM, vol 40, pp 226–230
3. Boud D, Feletti G (1998) The challenge of problem-based learning. Psychology Press
4. Brechner E (2003) Things they would not teach me of in college: what microsoft developers learn later. In: Companion of the 18th annual ACM

SIGPLAN Conf. on Object-oriented programming, systems, languages, and applications, ACM, pp 134–136
5. Bruegge B (1992) Teaching an industry-oriented software engineering course. In: Sledge C (ed) Software Engineering Education, Lecture Notes in Computer Science, vol 640, Springer Berlin Heidelberg, pp 63–87, DOI 10.1007/3-540-55963-9_40, URL http://dx.doi.org/10.1007/3-540-55963-9_40
6. Carrington D, Kim SK (2003) Teaching software design with open source software. In: Frontiers in Education, 2003. FIE 2003 33rd Annual, IEEE, vol 3, pp S1C–9
7. Dagenais B, Ossher H, Bellamy RK, Robillard MP, De Vries JP (2010) Moving into a new software project landscape. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, ACM, pp 275–284
8. Deak A, Sindre G (2014) Analyzing the importance of teaching about testing from alumni survey data. Norsk informatikkonferanse (NIK) 2013
9. DeMarco T, Lister T (1987) Peopleware: Productive people and teams. Dorseth House
10. Arie van Deursen EB Alex Nederlof (2013, retrieved: 24.08.2015) Teaching software architecture: with github! http://avandeursen.com/2013/12/30/teaching-software-architecture-with-github/
11. Edwards SH (2003) Rethinking computer science education from a test-first perspective. In: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM, pp 148–155
12. Edwards SH (2003) Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. In: Proceedings of the International Conference on Education and Information Systems: Technologies and Applications EISTA, vol 3
13. Elbaum S, Person S, Dokulil J, Jorde M (2007) Bug hunt: Making early software testing lessons engaging and affordable. In: Software Engineering, 2007. ICSE 2007. 29th Int. Conf. on, IEEE, pp 688–697
14. Fagerholm F, Johnson P, Guinea AS, Borenstein J, Munch J (2013) Onboarding in open source software projects: A preliminary analysis. In: Global Software Engineering Workshops (ICGSEW), 2013 IEEE 8th Int. Conf. on, IEEE, pp 5–10
15. Fagerholm F, Guinea AS, Münch J, Borenstein J (2014) The role of mentoring and project characteristics for onboarding in open source software projects. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, p 55
16. Garousi V, Felderer M (2016) Developing, verifying, and maintaining high-quality automated test scripts. IEEE Software 33(3):68–75
17. Glaser B, Strauss A (1967) The Discovery of Grounded Theory: Strategies for Qualitative Research. Observations (Chicago, Ill.), Aldine de Gruyter
18. Glass RL, Collard R, Bertolino A, Bach J, Kaner C (2006) Software testing and industry needs. IEEE Software 23(4):55–57

19. Janzen DS, Saiedian H (2006) Test-driven learning: intrinsic integration of testing into the cs/se curriculum. In: ACM SIGCSE Bulletin, ACM, vol 38, pp 254–258
20. Johnson M, Senges M (2010) Learning to be a programmer in a complex organization: A case study on practice-based learning during the onboarding process at google. Journal of Workplace Learning 22(3):180–194
21. Jones EL (2001) An experiential approach to incorporating software testing into the computer science curriculum. In: Frontiers in Education Conf., 2001. 31st Annual, IEEE, vol 2, pp F3D–7
22. Jones EL, Chatmon CL (2001) A perspective on teaching software testing. In: Journal of Computing Sciences in Colleges, Consortium for Computing Sciences in Colleges, vol 16, pp 92–100
23. Kaner C, Padmanabhan S (2007) Practice and transfer of learning in the teaching of software testing. In: Software Engineering Education & Training, 2007. CSEET'07. 20th Conf. on, IEEE, pp 157–166
24. Lübke D, Flohr T, Schneider K (2004) Serious insights through fun software-projects. In: Dingsøyr T (ed) Software Process Improvement, Lecture Notes in Computer Science, vol 3281, Springer Berlin Heidelberg, pp 57–68, DOI 10.1007/978-3-540-30181-3_6, URL http://dx.doi.org/10.1007/978-3-540-30181-3_6
25. Lustosa Neto V, Coelho R, Leite L, Guerrero DS, Mendonça AP (2013) Popt: a problem-oriented programming and testing approach for novice students. In: Software Engineering (ICSE), 2013 35th International Conference on, IEEE, pp 1099–1108
26. Marrero W, Settle A (2005) Testing first: emphasizing testing in early programming courses. In: ACM SIGCSE Bulletin, ACM, vol 37, pp 4–8
27. Ng S, Murnane T, Reed K, Grant D, Chen T (2004) A preliminary survey on software testing practices in australia. In: Software Engineering Conf., 2004. Proc.. 2004 Australian, IEEE, pp 116–125
28. Pham R, Singer L, Liskin O, Figueira Filho F, Schneider K (2013) Creating a shared understanding of testing culture on a social coding site. In: Int. Conf. on Software Engineering (ICSE), IEEE Press, pp 112–121
29. Pham R, Kiesling S, Liskin O, Singer L, Schneider K (2014) Enablers, Inhibitors, and Perceptions of Testing in Novice Software Teams. In: to appear: 22th International Symposium on the Foundations of Software Engineering (FSE 2014), Hong Kong, China
30. Pham R, Mörschbach J, Schneider K (2015) Communicating Software Testing Culture through Visualizing Testing Activity. In: 7th International Workshop on Social Software Engineering (SSE 2015), ACM, SSE 2015, URL http://dx.doi.org/10.1145/2804381.2804382
31. Pham R, Stoliar Y, Schneider K (2015) Automatically Recommending Test Code Examples to Inexperienced Developers. In: Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2015), ACM, ESEC/FSE 2015, URL http://dx.doi.org/10.1145/2786805.2803202

32. Radermacher A, Walia G, Knudson D (2014) Investigating the skill gap between graduating students and industry expectations. In: Companion Proceedings of the 36th International Conference on Software Engineering, ACM, pp 291–300
33. Rajlich V (2013) Teaching developer skills in the first software engineering course. In: Software Engineering (ICSE), 2013 35th International Conference on, IEEE, pp 1109–1116
34. Rogers EM (2003) Diffusion of Innovations, 5th edn. Free Press
35. Shepard T, Lamb M, Kelly D (2001) More testing should be taught. Communications of the ACM 44(6):103–108
36. Sim SE, Holt RC (1998) The ramp-up problem in software projects: A case study of how software immigrants naturalize. In: Software Engineering, 1998. Proc. of the Int. Conf. on, IEEE, pp 361–370
37. Singer L, Figueira Filho F, Cleary B, Treude C, Storey MA, Schneider K (2013) Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators. In: Proc. 2013 Conf. Comput. Supported Cooperative Work, ACM, NY, USA, CSCW '13, pp 103–116, DOI 10.1145/2441776.2441791, URL `http://doi.acm.org/10.1145/2441776.2441791`
38. Sowe SK, Stamelos I, Deligiannis I (2006) A framework for teaching software testing using f/oss methodology. In: Open Source Systems, Springer, pp 261–266
39. Spacco J, Pugh W (2006) Helping students appreciate test-driven development (tdd). In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, ACM, pp 907–913
40. Steinmacher I, Wiese IS, Conte T, Gerosa MA, Redmiles D (2014) The hard life of open source software project newcomers. In: Proc. of the 7th Int. Workshop on Cooperative and Human Aspects of Software Engineering, ACM, pp 72–78
41. Stuart HC, Dabbish L, Kiesler S, Kinnaird P, Kang R (2012) Social transparency in networked information exchange: a theoretical framework. In: Proc. of the ACM 2012 Conf. on Computer Supported Cooperative Work, ACM, New York, NY, USA, CSCW '12, pp 451–460, DOI 10.1145/2145204.2145275
42. Stuart HC, Dabbish L, Kiesler S, Kinnaird P, Kang R (2012) Social transparency in networked information exchange: a theoretical framework. DOI 10.1145/2145204.2145275
43. Whittaker JA, Arbon J, Carollo J (2012) How Google tests software. Addison-Wesley
44. Zagalsky A, Feliciano J, Storey MD, Zhao Y, Wang W (2015) The emergence of github as a collaborative platform for education. In: Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, CSCW 2015, Vancouver, BC, Canada, March 14 - 18, 2015, pp 1906–1917, DOI 10.1145/2675133.2675284, URL `http://doi.acm.org/10.1145/2675133.2675284`