Running a Red Light: An Investigation into Why Software Engineers (Occasionally) Ignore Coverage Checks

Alexander Sterk  
a.jh.sterk@gmail.com  
Delft University of Technology  
The Netherlands  

Mairieli Wessel  
mairieli.wessel@ru.nl  
Radboud University  
The Netherlands  

Eli Hooten  
eli.hooten@sentry.io  
Sentry.io  
United States of America  

Andy Zaidman  
a.e.zaidman@tudelft.nl  
Delft University of Technology  
The Netherlands  

ABSTRACT

Many modern code coverage tools track and report code coverage data generated from running tests during continuous integration. They report code coverage data through a variety of channels, including email, Slack, Mattermost, or through the web interface of social coding platforms such as GitHub. In fact, this ensemble of tools can be configured in such a way that the software engineer gets a *failing status check* when code coverage drops below a certain threshold. In this study, we broadly investigate the opinions and experience with code coverage tools through a survey among 279 software engineers whose projects use the Codecov coverage tool and bot. In particular, we are investigating why software engineers would ignore a failing status check caused by drop in code coverage. We observe that >80% of software engineers — at least sometimes — ignore these failing status checks, and we get insights into the main reasons why software engineers ignore these checks.

KEYWORDS

software testing, code coverage, coverage checks

1 INTRODUCTION

Software has become a critical aspect of modern society as we are more and more relying on software for everyday tasks. Because of our reliance on software, its quality and reliability is indispensable [4, 24]. The repercussions of unreliable and incorrect software can be severe, ranging from users getting frustrated, over huge financial losses [31, 32], or years of lost research [33], to causing injury or even death [26]. As Aniche et al. state “Making sure software works is maybe the greatest responsibility of a software developer” [2].

Software testing is essential to ensure the quality of the software systems that we as a society rely on [5, 6]. However, writing tests is a tedious and time consuming task [1, 3, 5]. Code coverage, which refers to the percentage of code that is executed by a test suite, is both an important metric in assessing the reach of a test suite, but also an indicator of areas of code that may need further testing [47].

One of the more popular ways to measure code coverage in the development process is using code coverage tools or bots, which are integrated into popular open-source development platforms, such as GitHub, GitLab, or Bitbucket. These tools collect coverage data from running tests, calculate various coverage metrics, and report the results back to the developers [12, 44].

In this paper, we explore the developers’ rationale when considering code coverage and working with a code coverage tool, specifically the Codecov tool. Codecov integrates with many different open-source development platforms and supports a large number of programming languages. It generates coverage reports for each commit and in the case of a pull request, posts a comment to the pull request that provides a summary of the full report. It can provide a “status check” for pull requests, which can either be a pass or a failure, if the coverage results of the pull request do not pass the standards of the project. It can also report the coverage results through email or Slack. An example of a comment and the status checks is given in Figure 1. Important for our particular study, is that Codecov also calculates coverage levels for individual pull requests, which Codecov call *patch coverage*. This only measures coverage for the lines that were actually changed in the pull request2.

Due to Codecov being free for students and public projects, and because it is used by over a million software developers, they have a large amount of available coverage data, which they made available to us, to use for research purposes. Specifically, that data made us wonder why software engineers would deliberately ignore indicators by the coverage tool. As such, our guiding research question is the following:

**RQ:** Why are coverage checks ignored by software engineers?

In this paper, we explore the developers’ rationale when considering code coverage and working with a code coverage tool, specifically the Codecov tool. We do so through a survey among
279 software engineers of whom the projects make use of Codecov. Our findings indicate that >80% of developers sometimes purposefully ignore a coverage check, for a variety of reasons.

2 STUDY SETUP

The goal of our study is to explore how developers use and look towards code coverage in open-source development projects. Specifically, we aim to come up with reasons why coverage checks would get ignored, as well as categorize what developers would consider good coverage practices. To address a sizeable population, we have chosen to conduct a survey to gauge the aforementioned elements. When preparing and executing the survey, we have followed the guidelines set by the Delft University of Technology’s Research Ethics Council and received approval from that council to conduct our study. We used a GDPR compliant version of Qualtrics for our survey.

2.1 The Survey

Mindful of the different roles that exist within open-source software development [16, 17], we have opted to use two different paths in the survey based on whether a respondent identifies themselves as a maintainer, a software engineer that manages, reviews, and integrates contributions (pull requests), or a contributor, a software engineer that makes contributions and opens a pull request. Both groups mostly received the same questions, although some were written from the perspective that better aligns with the chosen group.

During a pilot run with 4 PhD students in the research group, we have received feedback on the survey, which can be summarised as follows: (1) the survey was considered lengthy, (2) some ordinal scale options were questionable or difficult to choose between, (3) a lack of images and clarity in the survey when describing different functionalities of code coverage tools. This feedback was incorporated in the final survey design. A summary of the final survey is shown in Table 1.

2.2 Recruitment of Participants

The target audience for our survey were developers familiar with using code coverage tools on open-source development platforms. We have been fortunate enough to be able to rely on Codecov for obtaining users which had contributed to GitHub projects that use Codecov. The information we collected was the number of contributions, the time of the last contribution, usernames and repository names. The entire dataset consisted of around 260,000 entries (including duplicates).

We then used GitHub’s API to look up these usernames and find user profiles with public email addresses. This was very important since we did not want to breach any terms of service or be a nuisance to people. Subsequently, we were left with roughly 90,000 email addresses. From these we filtered users based on the number of commits they have made overall, and when their last commit took place. Only users with over 100 commits, and a last commit between the start of 2019 and the end of 2021 were kept. This was done to (1) prevent absolute newcomers from taking the survey, and (2) assure that participants had recent experience with the tool. This filtering resulted in a list of 11,000 people, of which we randomly selected 2,000 to send our survey to, by email. We opted for 2,000, to account for a potentially low response rate of 6%, which was a rough estimate based on previous studies in software engineering [37, 43].

2.3 Deployment of the Survey

In December 2021, our survey was sent out to 2,000 email addresses. During this time, the survey was open and able to receive submissions for roughly a month. However, most of the responses were submitted in the first two weeks. We obtained 379 response attempts in total, and 278 complete responses of respondents who finished the entire survey, i.e., a response rate of ∼14%.

3 RESULTS

This section describes the results from the survey. We only report on the 278 surveys that were completed. Our replication package contains all anonymized data [39].
Table 1: Summary of the survey questions. The final column indicates whether this question was dependent on a previous question, or asked to All respondents, or only Maintainers, or Contributors.

<table>
<thead>
<tr>
<th>#</th>
<th>Question</th>
<th>Type</th>
<th>A/M/C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demographic questions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>For how many years have you been developing software? You can consider all hobby, study and/or work experience.</td>
<td>Open</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>For how many years have you been active on open-source development platforms, such as GitHub, Gitlab, etc?</td>
<td>Open</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>How often do you contribute to an open source project?</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>3a</td>
<td>On average, I make a contribution (e.g. a commit, a pull request, etc) to somebody else’s project(s) ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3b</td>
<td>On average, I make a contribution to my own project(s) ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3c</td>
<td>On average, I review other people’s contributions ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>When contributing to open source projects, I primarily act as a: (Code contributor OR Maintainer/Project Manager)</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>Do you work in software development in a professional capacity? (Yes OR No)</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>What is your job title?</td>
<td>Open</td>
<td>Q5=&quot;Yes&quot;</td>
</tr>
<tr>
<td>7</td>
<td>How long have you been performing the following tasks, in either a professional or hobby capacity?</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>7a</td>
<td>Automatic software testing tasks, such as writing unit tests or integration tests</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7b</td>
<td>Manual software testing tasks or performing any sort of manual quality assurance functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7c</td>
<td>Performing code review of others’ contributions to any project, either open or closed source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>When it comes to automated software testing (e.g. unit testing, integration testing, etc) and its relationship to overall code quality, do you believe that automated software testing is: (Likert scale of importance)</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>Main questions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>How often do you use code coverage tools outside of GitHub? For example, on your own machine.</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>How often do you utilise the information from code coverage tools on GitHub?</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>10a</td>
<td>I use code coverage tools while developing/contributing ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10b</td>
<td>I use code coverage tools while reviewing ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>In the last question you answered you never utilise the information from code coverage tools on GitHub. Do you have any particular reason why you do not use code coverage tools on GitHub? After this question, the survey ends.</td>
<td>Open</td>
<td>Q10=&quot;Never&quot;</td>
</tr>
<tr>
<td>12</td>
<td>In your experience, what is a good coverage goal for a project? For example, is there a certain set of rules you’d like to follow, or a certain target you’d like to reach? If it’s possible, please also give us your reasoning.</td>
<td>Open</td>
<td>A</td>
</tr>
<tr>
<td>13</td>
<td>Please give your opinions on the following statements:</td>
<td>Closed</td>
<td></td>
</tr>
<tr>
<td>13a</td>
<td>Code coverage is a good metric to consider as part of overall code quality</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>13b</td>
<td>Code coverage tools on open-source platforms provide an incentive to improve coverage and/or write tests.</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>13c</td>
<td>I am more likely to approve a pull request that improves code coverage than ones that lower it.</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>13d</td>
<td>If my pull request improves coverage, it is accepted more quickly, in my experience.</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>14</td>
<td>How often do you write tests for projects you are contributing to?</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>15</td>
<td>How often do you write a test or multiple tests with (just) the intent to improve the code coverage?</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>16</td>
<td>How often are you asked/encouraged to better test your contributions, in the comments of a pull request you opened?</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>16a</td>
<td>People ask me this ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16b</td>
<td>A coverage tool asks me this ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Do you remember a particularly interesting instance where this (see Q16) happened? How did the situation get resolved?</td>
<td>Open</td>
<td>C</td>
</tr>
<tr>
<td>18</td>
<td>How often do you have to tell a contributor that their tests need to be improved, based on the results of a coverage tool?</td>
<td>Closed</td>
<td>M</td>
</tr>
<tr>
<td>19</td>
<td>Do you remember a particularly interesting instance where this (see Q16) happened? How did that situation get resolved?</td>
<td>Open</td>
<td>M</td>
</tr>
<tr>
<td>20</td>
<td>The following actions constitute incentives to improve coverage and/or write tests, and can also be done by code coverage tools. Please rank them on how much incentive you think they provide, from most incentive to least incentive.</td>
<td>Ranking</td>
<td>A</td>
</tr>
<tr>
<td>20a</td>
<td>Leaving a comment on a pull request, summarising the coverage changes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20b</td>
<td>Giving a failing status check for a commit or pull request, preventing automatic merging</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20c</td>
<td>Annotating uncovered lines in the “Files changed” overview of a pull request</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20d</td>
<td>Notifying users through messaging applications or email, if coverage is lowered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20e</td>
<td>Reminding users of contributing guidelines, when opening a pull request.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>In your experience, what is the best way to provide incentive for improving code coverage?</td>
<td>Open</td>
<td>A</td>
</tr>
<tr>
<td>22</td>
<td>Can you come up with situations where you would ignore a failing coverage check? What are your reasons?</td>
<td>Open</td>
<td>A</td>
</tr>
<tr>
<td>23</td>
<td>How often do you neglect or ignore a failing coverage check on a commit or pull request?</td>
<td>Closed</td>
<td>A</td>
</tr>
<tr>
<td>23a</td>
<td>I ignore a failing coverage check when contributing to a project ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23b</td>
<td>I ignore a failing coverage check when reviewing a pull request ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Can you give us 2 things you like about using a code coverage tool on an open-source platform?</td>
<td>Open</td>
<td>A</td>
</tr>
<tr>
<td>25</td>
<td>Can you give us 2 things you dislike about using a code coverage tool on an open-source platform?</td>
<td>Open</td>
<td>A</td>
</tr>
</tbody>
</table>

3.1 Demographic

General demographics. The demographics of our survey population can be seen in Figure 2. We observe that a majority of the participants have several years of experience with software development. Furthermore, while most participants actively work in a software development field, there are a substantial number of participants that do not. These participants come from a number of different fields, such as biology, mathematics, ecology, mobile development, etc. Some participants work in research, while others work in a more practical setting. Furthermore, some participants work in a high position, such as CEO, while others have listed themselves as interns. We clustered the different occupations into larger categories, which are shown in Table 2.

Overall, the participants to our study seem quite diverse, and coupled with the fact that our participants are from a set of diverse organisations, this should ensure the generalizability of our insights.
Open-source experience. Figure 3 shows that most participants have experience with open-source development platforms. We also observe the frequency at which they perform certain tasks related to open-source software development on these platforms. However, we also noticed some inconsistencies, with some participants claiming to have over 20 years of experience, while platforms such as GitHub have not been around as long. Nonetheless, it is entirely possible that participants have experience with older version control systems. Furthermore, we find that some participants rarely or never perform one or more of these tasks.

When gauging whether a participant to the survey considers themselves more of a code contributor, or more of a project maintainer, we found the groups to be of almost equal size (143 maintainers vs. 135 contributors). Depending on the role that they assigned to themselves (maintainers vs. contributor), some of the follow-up questions in the survey will differ.

3.2 Quantitative Results

Since the quantitative questions are all closed or Likert scale questions, their results can all be interpreted using graphs.

Code coverage tool usage. The first set of questions is regarding how often the participants use code coverage tools. From Figure 4 we find that using code coverage tools for each pull request is the most popular answer, regardless of whether one is contributing to a pull request, or reviewing one. In the case that a participant responded “Never” to both questions, this would mean that they lack the experience that is required for the remainder of the survey. Therefore, we would have them skip the remainder of the survey. This occurred 9 times, causing the number of responses for the remaining questions to go down to 269.

The right-most graph of Figure 4 shows a quite diverse answer to whether participants use code coverage tools outside of GitHub, for example on their local device while developing.

General attitude towards coverage and testing. When asked about the general necessity of automated software testing and code coverage, most participants responded that they consider testing and code coverage as important, as can be seen in Figure 5. However, not all respondents feel this way, as one person indicated that automated software testing is not at all important.

Frequency of writing tests. Figure 6 shows the results for two questions asking how often the participants write tests. An interesting observation is that 0 respondents answered that they never write tests. Even the one participant who answered that automated software testing is not important at all still writes tests. Furthermore, we also find that “For each pull request” is the most popular answer for the first question. Interestingly, the second graph indicates that respondents frequently write tests with the purpose of improving coverage.

Coverage and incentive. Keeping in mind that the middle graph of Figure 6 indicates that most participants write tests with the sole purpose of improving coverage, we wonder about the potential incentive that code coverage tools provide for writing tests and/or improving the coverage metric(s) of a code base. When we consider the right-most graph in Figure 6, we see that almost all participants agree, either somewhat or strongly, that code coverage tools provide an incentive to improve coverage, with no big difference of opinion between the contributors and the maintainers.

Secondly, we asked contributors and maintainers near identical questions (Q13c and Q13d in Table 1) regarding the acceptance rate of pull requests, based on their coverage levels. In Figure 7 we observe a difference of opinion between what contributors believe,
versus what maintainers claim: maintainers indicate they are more likely to accept a pull request which improves coverage, while contributors indicate to think that a coverage-improving pull request will not get accepted more quickly compared to other pull requests.

We also asked participants to rank different features of code coverage tools, from what they consider providing the most to the least incentive to improve coverage. In Figure 8 we graph both the number of times a feature has been put in first position, as well as its average rank, with 1 being the highest, and 5 being the lowest. The **status check** is by far ranked as the most incentivising feature. Nevertheless, every other feature has also been ranked first by one or more participants at some point. We observe that contributors and maintainer share the same opinion.

**Neglecting the coverage check.** Figure 9 shows the frequency with which the participants ignore a failing coverage check when both contributing and reviewing. Of interest to observe is that >80% of the respondents across categories indicate to at least sometimes ignore failing coverage checks. Additionally, we see that maintainers tend to ignore failing coverage checks slightly more frequently.

**Pointing out decreasing coverage.** Figure 10 provides insight into how often the respondents are asked to improve coverage, or need to ask other contributors to improve coverage.

A first observation is that a large group of contributors indicate that they have never been asked to improve their coverage by a human, and an even larger group says they have never been asked by a coverage tool. Secondly, it appears that overall coverage tools are perceived to ask to improve coverage more frequently than humans. Thirdly, when we compare maintainers to contributors, we see very similar numbers, especially for the "By a human" bars, with the exception for the "A few times a month" option. A possible explanation could be that maintainers review many pull requests from different contributors per month, but contributors might not have their pull requests reviewed by many different reviewers per month, i.e., maintainers overall review more varied pull requests.

### 3.3 Qualitative Insights

For our analysis of the qualitative (open) questions, we applied open and axial coding [40] throughout multiple rounds of analysis. We started by manually assigning each answer one or more codes; if a response was unclear and could not be given a code, it was discarded. To check for consistency and agreement in our coding process, the first and second authors performed the open coding of the first 25 participants’ answers separately. Then, these two researchers discussed the emergent codes and reached a negotiated agreement in a hands-on meeting. After reaching an agreement, the first author coded the remaining answers.

Afterwards, we performed axial coding by grouping the codes into larger categories of responses. For each question, we listed the codes that were generated from the open coding process and assigned them into groups based on how closely the codes were related to one another or a particular topic. In the case a code could not be properly categorized, it was added to a "Miscellaneous" category. We then retrofitted the categories back to the original responses, based on which codes they had. Finally, we counted how many responses belonged to each category. The axial coding was performed by the first author, and discussed with the second and third authors until reaching consensus. The coding process was conducted using a continuous comparison technique [15], wherein we continuously compared emerging codes with existing findings to validate our interpretation. The full list of responses and codes can be found in our online replication package [39].

As mentioned above, we counted the number of responses per category, to find how many participants mentioned a specific category in their responses to each question. It is possible for a response to mention multiple categories, and therefore count for multiple
How often do you write tests when contributing?

How often do you write tests with the intent of improving coverage?

Code coverage tools on open-source platforms provide an incentive to improve coverage.

For contributors: if my pull request improves coverage, I feel it is accepted quicker.

For maintainers: I am more likely to approve a pull request that improves coverage.

For contributors: how often are you encouraged to improve your coverage?

For maintainers: how often do you (have to) ask a contributor to improve their coverage?

Best way to incentivize for coverage. Previously, we asked respondents to rank different features of coverage tools, based on how much incentive they provide. For this question however, participants were able to give their own input on the best way to encourage writing tests and improving coverage. The results can be found in the 2nd part of Table 3.

We observe that participants most often list negative impacts as the best way to provide incentive to improve coverage. This typically comes down to blocking the merge of the pull request, or closing it all together. One caveat with these responses is that the more popular answers (negative impacts, using PR comments, getting a coverage report) are features that coverage tools already

interest when looking at the first question in Table 3, is that for each category of responses the ratio of participants that mention it is (almost) the same for both contributors and maintainers. Secondly, nearly 50% of all respondents mention some kind of high coverage goal in their responses. However, for both groups at least 20% of the participants believe that striving to achieve some arbitrary target is not the right goal to set.

Good coverage goal. The first open question we asked to the participants, was for them to describe their own idea of a good coverage goal to set for a project. The first thing that piques our

categories. Furthermore, the results are grouped per participants’ roles as either a contributor or a maintainer again. These results can be found in Tables 3 through 5. We will now go through them, one question at a time.

Good coverage goal. The first open question we asked to the participants, was for them to describe their own idea of a good coverage goal to set for a project. The first thing that piques our

Figure 6: Frequency of writing tests and code coverage improvement

Figure 7: Beliefs about pull request acceptance and code coverage

Figure 8: Ranking the most incentivising feature of code coverage tools

Figure 9: Frequency of ignoring a coverage check

Figure 10: Frequency of asking to improve coverage
we already mentioned them previously in the survey. Therefore, it becomes hard to interpret, or offer no guidance on what to expect barrier and easy set-up for coverage tools, while in the dislikes list of contributors would not have to set them up themselves. Moreover, this is also true for the tooling got too much in the way, it took too much time, and I decided to focus on real problems instead. The largest overall expressed dislike by both groups, is that code quality testing is more sensible matter, where uncovered items are justified properly, a few years ago I spent a lot of time writing coverage tests. Then I gave up: the tooling got too much in the way, it took too much time, and I decided to focus on real problems instead. We would expect this, since maintainers might be more involved with and interested in the overall state of a project’s coverage throughout its development. Similarly, this would also hold true for the “third-party host concerns”. Another dislike is that a decent percentage of contributors mention the difficult setup as something they dislike. This is something we would expect from maintainers, but not from contributors, since the whole point of code coverage tools is to have them on something like GitHub, running during the CI pipeline. This means that contributors would not have to set them up themselves. Moreover, in the list of likes we find that 10% of both groups enjoy the low entry barrier and easy set-up for coverage tools, while in the dislikes table we find that there is another set of respondents that claim the exact opposite.

The largest overall expressed dislike by both groups, is that code coverage tools lead to people treating code coverage as a form of vanity metrics like % code coverage don’t convince people that don’t care about testing; not getting paged due to bugs does. The second largest
dislike is that the results of the coverage tool can be incorrect, irrelevant, or too small to really matter [18].

Why are coverage checks ignored. Finally, we move on to one of the most important questions of the survey. We would like to find out in what situations the participants would ignore a failing coverage check. We asked this to the participants in the form of an open question, and the results can be found in Table 5.

Firstly, it seems coverage failures tend to be ignored when they are minimal, e.g., less than 1% coverage delta. Something like this can happen when only a few lines of code are added, or perhaps deleted, or by introducing an extra branch, or removing a test case, which happens on occasion [36]. Out of all the reasons, this one is given most often by the maintainers.

Secondly, the coverage failure can be a result of another failure, for example in the CI pipeline. Another reason that is given is that the coverage information is simply incorrect, or unrelated to the PR in question.

The third reason is priority. Some issues need to be merged quickly, as they contain fixes for critical bugs, or very desirable automation and the ease of adding it to new product.*

Table 4: Likes and dislikes of coverage tools according to the respondents.
4 DISCUSSION & IMPLICATIONS

4.1 Revisiting the Research Question

Through the responses to our survey, we have found indications that failing coverage checks are sometimes ignored. Overall, we have observed that >80% of the respondents have indicated that they ignore failing coverage checks at some point. We also uncovered four key reasons for ignoring coverage checks, namely: (1) a small coverage delta, (2) coverage calculation failures or mistakes, (3) priority lies with production code [30, 45, 46], and finally, (4) the difficulty of testing complex code [14, 41].

In what follows, we try to formulate recommendations for open-source developers and developers of code coverage tools to potentially alleviate some of the aforementioned issues.

4.2 Recommendations for Open-Source Developers

One of the main reasons for ignoring a failing coverage check is that the failure is due to a minimal decrease in coverage. One way to alleviate this problem is by configuring their coverage tools with a certain threshold, so only larger decreases trigger an actual failure. Furthermore, developers can also configure the coverage tools to decrease the noise they create. A practical approach would be to group multiple bot comment reports into a single report, which was demonstrated as useful in a paper by Wessel and Steinmacher [44].

Another big reason for neglecting fixing the coverage is simply the high complexity of testing (certain parts of) the code. This is a call to arms to developers to ensure that the code is testable [11], and potentially refactored in a test-driven way [34].

Another recommendation is writing clear guidelines for contributing, but with a clear goal for testing and/or coverage targets. The answers to the question “What is a good coverage goal?” in Table 3 can provide some inspiration for contribution guidelines.

4.3 Recommendations for Code Coverage Tool Developers

The dislikes listed in Table 4 provide developers of code coverage tools with a list of opportunities to improve their tools. Some of the problems that are mentioned by both maintainers and contributors are: unclear or wrong results, a difficult setup process, and noise.

Zooming in specifically on the matter of robustness of results, it might be worthwhile to investigate ways to ignore individual methods or lines that cause problems, e.g., through annotations in source code. Another angle here is to annotate flaky tests, so that developers better understand where fluctuations in code coverage is coming from (confer [42]).

A large reason for why failing coverage checks are ignored is because the contributors and/or maintainers of a project have other priorities, do not have enough time to immediately address the failure, or because they are working on experimental code, which might change a lot in the future. Therefore, we think it would

---

For example: https://docs.codecov.com/docs/codecov-yaml, last visited October 4th, 2023.
be beneficial if code coverage tools provided an optional integration with issue tracking systems (i.e., GitHub issues, Jira, etc.), to (automatically) open new issues for uncovered, but merged, code changes. From a developer's perspective this would make it easier to track the testing technical debt [29] with a system they are likely already familiar with. Similarly, an integration with a tool like TestKnight, that creates boilerplate code for tests yet to be written, could prove interesting [9]. Finally, a tool like TestAxis could be extended to report coverage information from cloud-based continuous integration services directly into the IDE [8].

4.4 Threats to Validity

Internal validity. One of the main threats to validity is the possible bias we introduced while categorising the qualitative responses. To mitigate this threat, we used a process in which the first responses for each open question were analysed by the first and second author, who then compared their categories. During this process, no major differences were found. Subsequently, the other responses were categorized by the first author, and later verified by the second author.

In order for participants to get acquainted with the idea of code coverage tools, we used some screenshots of functionality provided by Codecov. This could have unintentionally steered participants to think of Codecov in particular, when we asked them about their experiences with any tool. We tried to mitigate this threat by also making references to other possible code coverage tools, and consistently using the term "code coverage tool" in our questioning.

External validity. We were limited by both time constraints and GitHub’s API to invite and include all potential users of coverage bots for our survey. As such, we had to narrow our search by first querying people from Codecov’s database. This introduces a bias in our results, as while all our participants have used Codecov before, it does not mean that they have used other tools before. In future work, we intend to widen the scope of our investigation to other code coverage tools.

We randomly sampled 2,000 users of Codecov to invite them for our survey. We ended up with an almost even split of 135 contributors and 143 maintainers. This means that we could gather opinions and experiences from different perspectives and developer profiles. In future work, we intend to study more developers, who also use a variety of code coverage tools.

5 RELATED WORK

Ivanković et al. have studied code coverage challenges and experiences at Google [22]. Through a survey among 512 employees of Google they observe that "a substantial number of developers do use code coverage on a regular basis and find value in it". Their study also highlights how developers find it useful to obtain coverage information within the tools developers commonly use, something that relates to how the Codecov bot works on GitHub. Also similar to our study, they mention that errors in code coverage numbers are consuming time and energy from software engineers.

Elbaum et al. have established that even small changes in production code, can cause big fluctuations in code coverage values [13].

The respondents to our survey have indicated to regularly write additional test cases to improve the code coverage. The study by Kochar et al. puts this into perspective [27]. In particular, their findings are that a relationship between code coverage and post-release defects is non-existent or statistically insignificant. As such, they warn that designing test cases with the sole purpose of increasing coverage may or may not translate to higher bug finding rate. Our findings highlight that although coverage is commonly used as a yardstick for test adequacy, its impact should not be overestimated.

Beller et al. have investigated the reasons for failing continuous integration builds [7]. They have observed that the key reason for build failures are test failures. Instead, our work looks at failing status checks caused by missed code coverage targets.

Khatami et al. have looked into the awareness of software engineers about the quality assurance practices in vogue in the GitHub projects that engineers contribute to [25]. Through a large-scale survey they have established that ~80% of the respondents indicate that the coverage their project reaches is easy to retrieve, and that ~37% is aware of coverage targets.

Hilton et al. have studied code coverage evolution [20]. They found that measuring the patch coverage, like the Codecov tool that we study enables, provides visibility into the impact of the changes that software engineers make.

There are a number of other studies in the field of code coverage, but they are typically based on a few projects, based in a single language [19, 21, 35], or at a single corporate entity [22, 28]. The results of these studies might therefore not necessarily generalize well to open-source development. Furthermore, most of these studies measure the impact of code coverage on the development or testing processes, but not the impact on the developers themselves.

6 CONCLUSION & FUTURE WORK

In this study we have broadly investigated the perceptions and opinions of open-source software engineers on code coverage tools in general, and Codecov specifically. More specifically, we tried to establish why code coverage checks are ignored?. Through an online survey among 279 open-source software engineers of which the projects make use of Codecov, we find that ~80% of the software engineers have at some point ignored code coverage checks. We also find four key reasons for ignoring these checks, namely: (1) a small coverage delta for the particular commit or pull request, (2) a failure to compute code coverage, or wrong code coverage results, (3) a difference in priorities as production code is deemed more important, and finally, (4) the difficulty of testing complex code.

In future research, we aim to better understand the role of contribution guidelines on the process of working with code coverage tools. We will also look into the relationship between test flakiness and fluctuations in code coverage reports. In particular, we see opportunities to (1) better understand the influence of flaky tests on code coverage results, and (2) investigate whether excluding flaky tests from coverage calculations, e.g., through annotations in code, would help software engineers work better with coverage tools.

ACKNOWLEDGMENTS

This research was partially funded by the Dutch science foundation NWO through the Vici “TestShift” grant (No. VLC.182.032) and conducted as part of Alexander Sterk’s master thesis [38].