Software Quality Assurance Analytics: Enabling Software Engineers to Reflect on QA Practices

Ali Khatami
Delft University of Technology
Delft, The Netherlands
s.khatami@tudelft.nl

Carolin Brandt
Delft University of Technology
Delft, The Netherlands
c.e.brandt@tudelft.nl

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

***Abstract*—Software engineers employ a variety of approaches to ensure the quality of software systems, including software testing, modern code review, automated static analysis, build automation, and continuous integration. Previous research has shown that software engineers lack situational awareness about the quality assurance (QA) practices in their projects. We propose *software quality assurance analytics* to help software engineers become aware of their QA usage, and reflect upon it. We have developed a prototype quality assurance analytics dashboard coined `RepoInsights` that provides both a global overview and a comparative aspect of the different QA practices. Through a qualitative study involving 14 participants who have completed an assignment using our `RepoInsights` dashboard, we elicit their perspective on quality assurance analytics. We observe that the dashboard has enabled the software engineers to reflect on the QA practices of software projects. Additionally, we distill a set of recommendations for future quality assurance analytics.**

***Index Terms*—Software Quality Assurance, Software Analytics, Empirical Software Engineering, Software Testing, Code Review, Automation Workflows**

I. Introduction

Because of the growing importance of software in our society, the *quality assurance* (QA) of that software is indispensable [1]. This view is supported by countless reports of software failures that have caused severe harm to businesses, people, or society as a whole [2], [3]. Software engineers can use a range of QA mechanisms to ensure the quality of the software systems they produce, including software testing [2], [4]–[8], modern code review [5], [9], [10] automated static analysis [11]–[15], and build automation [16]–[19]. However, a recent study has indicated that software engineers lack situational awareness [20] when it comes to the QA efforts of the projects they are maintaining or contributing to [21].

An established approach to raise awareness is software analytics (SA), which makes software data available to engineers and managers, empowering them to gain and share insights leading to better decisions [22]. The combination of insights that 1) quality assurance is generally considered important, 2) the existence (and use) of a plethora of QA mechanisms, and 3) the lack of situational awareness of whether, how, and to what degree these QA mechanisms are used, leads to our idea to explore software analytics focused on QA data. We conjecture that providing so-called *quality assurance analytics* (QAA) to software engineers will increase their awareness about the quality assurance situation in their projects. This in turn can be useful to maximize the use of (a combination of) QA mechanisms, and hopefully lead to less quality incidents.

In this paper we investigate **how to design a quality assurance analytics dashboard in order to empower software engineers to reflect on the QA practices of their projects**. To this end, we design a QAA dashboard that provides a global overview of quality assurance-related activities in open-source software repositories and allows for comparisons with other projects. We develop a prototype of a QAA dashboard, `RepoInsights`, and evaluate it through a pretest-posttest study including semi-structured interviews with 14 software engineers from open-source, industry and academia. Our study is guided by the following research questions:

RQ1: How do software engineers apply QA practices and how do they reflect on them?
RQ2: What are software engineers' perspectives on a quality assurance analytics dashboard like `RepoInsights`?
RQ2.1: In what ways did the QAA dashboard help software engineers reflect on QA practices?
RQ2.2: In what ways could the QAA dashboard be improved to help software engineers reflect on QA practices?

During our study, we first explore how the particiants of our study currently follow and reflect on QA practices (**RQ1**). We let them judge the QA of two software projects with `RepoInsights` to observe their use of a QAA dashboard. Based on their satisfaction with the dashboard, their verbalized thoughts and the plethora of feedback they provided, we elicit software engineers' perspective on our QAA dashboard (**RQ2**) and characterize the ways in which `RepoInsights` succeeds in letting engineers reflect on QA (**RQ2.1**), and the ways in which the QAA dashboard can be improved (**RQ2.2**).

In summary, this study contributes:

- A novel, conceptual design of a quality assurance analytics dashboard, and our prototype `RepoInsights`
- Insights from software engineers on how they currently adopt and manage QA practices in their projects.
- Insights on the successful aspects of our design and recommendations on how to improve the dashboard, based on the feedback from software engineers.

II. RepoInsights: A Quality Assurance Analytics (QAA) Dashboard

In this paper, we introduce the concept of a software quality analytics dashboard, designed to enhance software engineers' situational awareness surrounding QA practices [23]. It gives a ***global overview*** of quality assurance practices by collecting information and giving metrics and details related to the state of testing, automated workflows [24], code reviews, and guidelines, in one place. It also provides detailed views showing specific information on, e.g., pull requests. Additionally, it incorporates a ***comparative aspect***, enabling users to view similar information and statistics across a larger set of repositories. The dashboard is intended for software engineers that contribute to or maintain a software project. It serves as a vehicle to reflect on how adequate their project currently applies QA practices.

A. The Global Overview & Comparative Aspect

The **global overview** is designed around two main concepts: (1) situational awareness and (2) the complementarity of quality assurance practices. *Situational awareness* theory, rooted in psychology, refers to "the continuous extraction and integration of information to form a coherent mental picture, and then use this information to direct future actions" [25] and is important for effective decision-making in many environments (e.g., health care, air traffic control, etc.) [26]. A prior study explored open-source developers' situational awareness [20] of quality assurance aspects within their projects. The study identified areas of deficiency and compared awareness across various quality assurance practices [21]. The complementarity of quality assurance practices highlights the fact that using them in conjunction can enhance the overall quality perspective while at the same time might also reduce quality assurance efforts [27], [28].

The **comparative aspect** of the dashboard lets software engineers evaluate the information provided for their project in comparison to *other* projects. It thus enables them to independently judge their project's QA by comparing metrics to the average of other popular and active software projects.

B. Repository Dashboard Page

The repository dashboard page on `RepoInsights` has four primary sections: Testing, Automated Workflows (AWs), Pull Requests (PRs), and Guidelines (see Figure 1). These four sections comprise the **global overview** of the project's quality assurance. Beneath each section, there is a collapsible segment that presents similar information averaged across a set of projects, providing the **comparative aspect** (④ and ⑥ in Figure 1).

The provided information, including metrics, statistics, and graphs for each section of the dashboard are as follows: **Testing**: displays available test coverage information ① (Codecov[1] and Coveralls[2]) and provides a link to a detailed page with

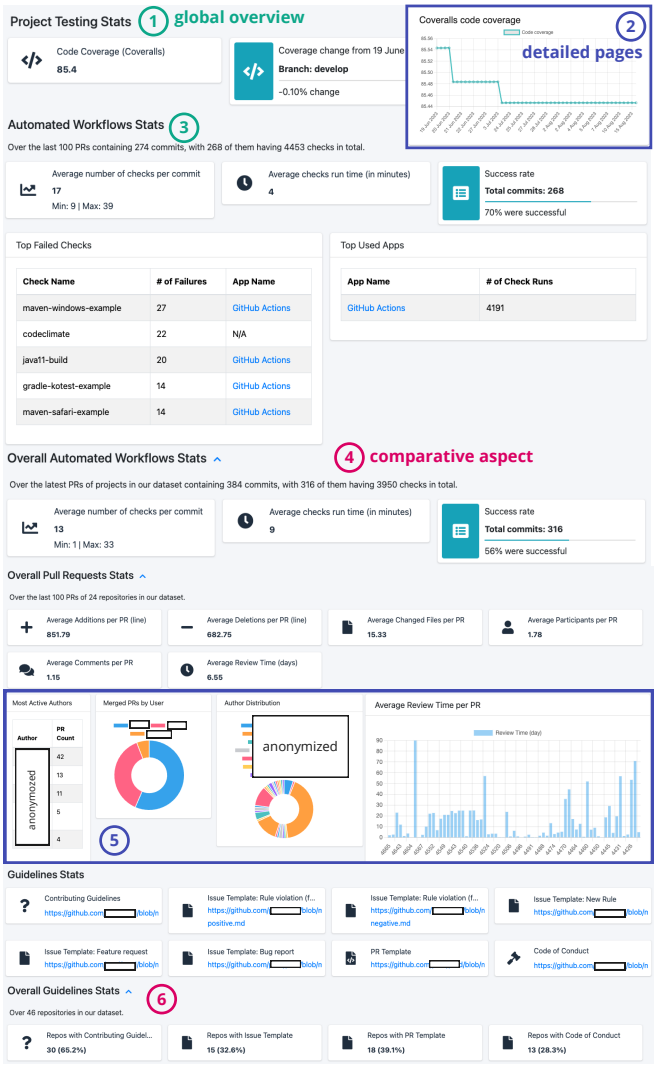


Fig. 1: An illustration showcasing various parts of the `RepoInsights` dashboard.

trend of test coverage over time ②. **Automated Workflows**: presents details regarding checks on commits of the latest 100 PRs ③, including: top failed checks (number of failures and their associated GitHub App handle), top-used GitHub Apps, success rate of checks, average checks runtime, and average number of checks per commit. **Pull Requests**: displays information on the last 100 PRs, including the average number of line additions and deletions, average number of files changed, average number of participants in PRs, and average review time. Additionally, it provides a link to a details page ⑤ with the following: top active authors, distribution of pull requests' authors and mergers, pull requests with the highest number of comments, and trend graphs for review time, line/file changes, and comments per pull request, along with a detailed table with pull request information. **Guidelines**: indicates the presence of contributing guidelines, code of conduct, and templates for issues and pull requests. If any of these guidelines exist, the dashboard page provides a link to them.

[1]https://about.codecov.io/, last visit: April 11, 2024
[2]https://coveralls.io/, last visit: April 11, 2024

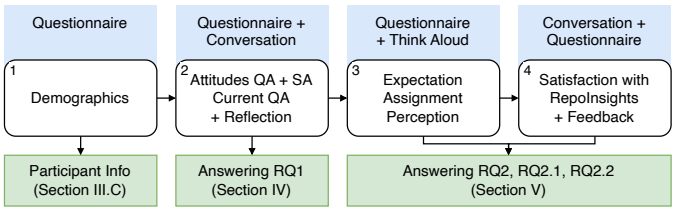


Fig. 2: Structure of our interview sessions.

C. Prototype Architecture Design

We adopt an engineering research approach, in which we develop a tangible prototype of a QAA dashboard. This prototype facilitates our study, enabling us to gather feedback on the concept of `RepoInsights` from software engineers.

The `RepoInsights` prototype follows a hybrid software architecture including a MERN[3] stack [29] for the app itself and a set of Python scripts that work as crawlers to collect the data through APIs of GitHub[4] (PR details, repository information, automated workflows, and guidelines), Coveralls[5], and Codecov[6] (test coverage information). The collected data is saved to MongoDB, retrieved by the MERN app to calculate statistics, and then display on the dashboard.

III. Study Setup

To evaluate our concept of a QAA dashboard, we follow a one-group pretest-posttest design with semi-structured interviews involving software engineers. We discuss how they currently reflect on quality assurance in their projects to get more insights on how a QAA dashboard can facilitate this current state (**RQ1**). To elicit concrete feedback on and suggestions for improvement for our design of a QAA dashboard, we developed the prototype `RepoInsights`. During the interviews, we ask the participants to judge two projects with `RepoInsights`, while thinking aloud. Then we ask them to evaluate different aspects of the tool, provide feedback, and sketch their potential usage. Through analyzing the participants' ratings and comments, we elicit software engineers' perspectives on a QAA dashboard like `RepoInsights` (**RQ2**). We are interested in the ways `RepoInsights` and our general concept of a QAA dashboard help participants in reflecting on QA practices (**RQ2.1**), as well as identifying the ways it can be improved (**RQ2.2**).

A. Study Design

We developed a questionnaire to guide our interview sessions. The steps of the interview align with the sections of the questionnaire and are shown in Figure 2. The questionnaire starts with informed consent and the participants' demographics, including questions about their experience in software development and whether they primarily work in open source, industry or academia (Step 1, Figure 2).

[3]MongoDB, Express.js, React.js, and Node.js.
[4]https://docs.github.com/en/graphql, last visit: April 11, 2024.
[5]https://docs.coveralls.io/api-introduction, last visit: April 11, 2024.
[6]https://docs.codecov.com/reference/overview, last visit: April 11, 2024.

In the next step, we ask our participants to rate statements regarding their attitudes towards quality assurance and software analytics. To ensure clarity, we also provide a definition along with examples for each concept. Understanding our participants' attitudes is important, as these could influence their evaluation of the tool they are about to interact with. Then, the interviewer asks about the concrete quality assurance practices they use in terms of testing, code reviews, automated workflows, and other guidelines. Additionally, the interviewer asks how they judge whether quality assurance practices are sufficiently used (Step 2, Figure 2).

The following step consists of a one-group pretest-posttest study design [30] with the participant interacting with the dashboard in between. This type of study is called quasi-experimental to indicate that it does not meet the scientific standards of experimental design [31], yet it allows reporting on facts of real user-behavior, even those observed in under-controlled, limited-sample experiences. First, we provide an explanation of our design of a QAA dashboard and ask the participants to rate statements regarding their expectations of such a tool (Step 3, Figure 2). After they complete the assignment with `RepoInsights`, we present the same set of statements and ask the participants to rate them based on their concrete experience with the dashboard. With this we gauge the participants' perception of the dashboard.

To encourage participants to engage with the dashboard, we devised a concrete task to solve during the assignment: *"assess two real-world open-source projects from GitHub across four quality assurance aspects: testing, automated workflows, pull requests, and guidelines."* Additionally, the participants were asked to judge the projects overall. The two projects, `fluentlenium/fluentlenium` and `pmd/pmd`, were carefully selected from our initial dataset of selected OSS projects on GitHub. This dataset included active and popular projects with ample data on pull requests (sampled using GitHub Search [32]), automated workflows, and available test coverage information. The two projects were chosen to have varied code review activity and contributor counts, enabling comparisons and observations of participants' reactions to different quality assurance levels. During the assessment, participants were encouraged to narrate their thoughts and provide reasons behind their judgments, with the interviewer actively prompting them for further explanations.

In the last section of the interview, we ask the participants to rate their satisfaction with the dashboard, provide feedback on how it can be improved and whether they would use such a QAA dashboard in their work (Step 4, Figure 2).

We obtained approval from our local ethics board and asked for informed consent to use and share the collected data for research purposes from the participants. For access to our full questionnaire, please refer to our provided artifact [33].

We conducted a pilot round of two interviews to estimate the interview duration and to identify any unclear questions. As a result, we increased the Likert scale range with 2 points for two questions and resolved an issue regarding the questionnaire's user interface. Subsequently, we requested the

Participant	Software Dev. Experience	First GitHub Contribution	Academia	Industry	Open Source	Primary Role Projects	University Education	Age range
P1	7 to 10 y.	3 to 7 y. ago	x	-	x	Maint.	PhD	25-34
P2	3 to 7 y.	1 to 3 y. ago	x	-	-	Maint.	PhD	25-34
P3	3 to 7 y.	3 to 7 y. ago	-	x	x	Maint.	BSc	18-24
P4	3 to 7 y.	3 to 7 y. ago	x	-	x	Maint.	BSc	18-24
P5	3 to 7 y.	3 to 7 y. ago	-	x	x	Maint.	MSc	25-34
P6	10+ y.	7 to 10 y. ago	-	-	x	Contrib.	-	35-44
P7	10+ y.	7 to 10 y. ago	x	-	-	Maint.	PhD	55-64
P8	10+ y.	7 to 10 y. ago	-	x	x	Maint.	-	45-54
P9	10+ y.	10+ y.	-	x	-	Maint.	BSc	35-44
P10	10+ y.	3 to 7 y. ago	x	-	-	Contrib.	PhD	55-64
P11	10+ y.	3 to 7 y. ago	-	x	x	Maint.	BSc	35-44
P12	10+ y.	3 to 7 y. ago	x	-	x	Maint.	PhD	35-44
P13	10+ y.	10+ y. ago	x	-	x	Maint.	PhD	25-34
P14	3 to 7 y.	7 to 10 y. ago	-	x	-	Maint.	PhD	25-34

TABLE I: Participants demographics.

pilot participants to review and update their responses to the modified questions, allowing us to incorporate their revised answers into the final data set.

B. Study Execution

We recruited 12 study participants through e-mails to software engineers who expressed interest in follow-up research after participating in a previous study. In addition, we recruited two more participants through convenience sampling in our professional networks to ensure a balanced representation of expertise in open-source software (OSS), industry, and academia.

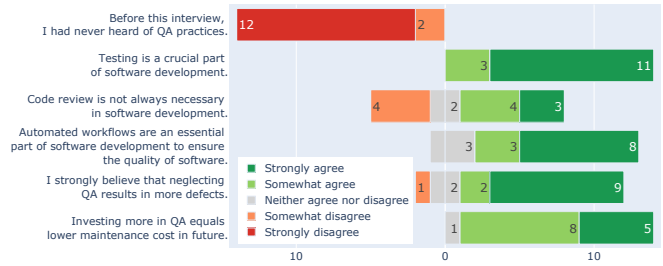
We conducted the interviews via Zoom sessions, granting participants remote access to the interviewer's computer to interact with both the questionnaire and the dashboard. We recorded the interviews to be able to transcribe them for the qualitative analysis. The first author, who conducted all 14 interviews, took notes during the interview, especially during the open conversation regarding the participants' current quality assurance practices (Step 3, Figure 2). We also surveyed participants to assess their satisfaction with the interview, including questions, assignments, and the overall experience. Responses indicated either a high or medium level of satisfaction.

C. Participants

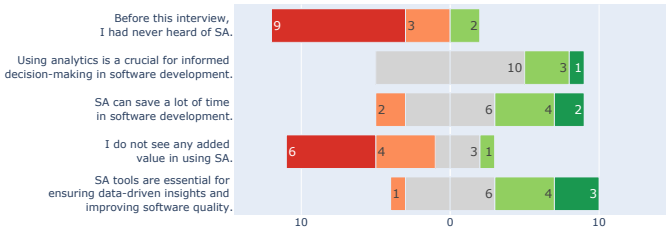
A summary of the participant demographics is in Table I. The majority of participants (85%) considered themselves to be in a maintainer role in the software projects they are involved in. Our participants represented a wide age range (∼20 to ∼60 years old) and mostly had education in computer science or related fields. They are often actively involved in OSS projects (64%), alongside working in academia or industry. One participant is exclusively working in OSS.

D. Data Analysis

The data analysis from the interviews was two-fold. We collected data from the questionnaire that the participants filled out during the interview, and applied open and axial coding [34] to analyze the recordings and transcriptions of the interviews. The first two authors started by analyzing one interview separately, and then discussed their experiences to



(a) Attitudes towards quality assurance practices.



(b) Attitudes towards software analytics.

Fig. 3: Attitudes regarding quality assurance practices and software analytics.

set guidelines for how to assign open codes to any interesting observations from the interview transcripts. Then each of the two analyzed half of the interviews separately before merging the codes together and grouping them into higher level codes. The authors discussed the emergent groups, employing constant comparison [35] with the original transcripts, until they reached a negotiated agreement [36]. These higher level codes, together with the data collected through the questionnaire, are the basis for our answers to the research questions. In the following section, each statement based on our codes is followed by parentheses clarifying which participants made statements that led to this observation. We publish the full codebook as part of our replication package [33].

IV. RQ1: How Do Software Engineers Apply QA Practices and How Do They Reflect on Them?

In this section, we summarize our insights from the first part of the interviews, executed as part of the pre-test. We asked about the participants' attitudes on QA practices, software analytics, and their current approaches for QA. We delved into how they assess the adequacy of their approaches and explored their motivations for extending QA efforts. Based on the high-level codes that arose during the open and axial coding of the interviews, we answer our first research question (**RQ1**).

A. Attitude Towards Quality Assurance and Software Analytics

Overall, our participants are aware of and positive towards the value of quality assurance. They are aware of software analytics but the majority does not make use of it in their projects, and they are neutral to positive about whether it is necessary for a successful project.

The answers to the questionnaire on QA are shown in Figure 3a. All participants were familiar with the term "quality assurance" prior to the interview. There was unanimous

agreement on the importance of testing. However, opinions differed regarding the necessity of code reviews, with half of the participants expressing either a moderate or high level of agreement that code reviews may not always be required. Concerning the impact of quality assurance, most participants held positive views about its ability to reduce defects and maintenance costs.

The participants had a mostly positive view of software analytics. They were neutral and leaning towards positive regarding its importance and necessity (Figure 3b). When asked if they used software analytics in their projects, five participants (36%) said yes. These participants gave examples of using custom scripts and internal tools for software analytics. Four participants reported checking these analytics daily, while one participant reported checking them every two weeks.

B. Well Justified Variety in Applied QA Practices

The most prominent insight from our interviews is the variety in adopted QA practices. This variety is so large that reporting the single practices goes beyond the scope of this paper. Nevertheless, we want to give a few examples.

With regards to adding tests, P2 explained that in their projects the responsibilities for testing lies with the individual developers and there is no particular enforcement of adding tests. In P11's project only the single maintainer adds tests to the project, while in P4's project the pull request reviewers are responsible to add tests. P13 shared that the level of testing in their projects strongly depends on how easy it is to test the application they are developing. While a library has defined inputs and outputs and can easily be unit tested, for applications that depend on a lot of other interfaces not under P13's control they often opt for "production debugging", i.e., seeing in production if something goes wrong and fixing it.

With regards to automated workflows, several participants mentioned using GitHub Actions (P3, P8, P9, P11, P12, P13, P14). Some combine this with other continuous integration services (P3, P11) or set up their own build infrastructure (P14). P3 mentioned that they have a custom solution for continuous integration, as *"It's nothing we can find on the market. So the framework is customized to our own need."*

When talking about code reviews, four participants (P1, P3, P5, P12) stressed the importance of following strict code review guidelines, insisting on never skipping them, even in urgent situations. On the other hand, P2 mentioned that they sometimes skip code reviews *"for very small PRs or for very small changes or fixes."*

Discussing contribution guidelines, we learned that while some projects have guidelines in the "contributing.md" file in the repository (P2, P3, P7, P9), others have them on the website of the project (P4), or keep the detailed guidelines only with their internal QA team (P3). There are also ways of communicating guidelines beyond documents, e.g., in meetings (P1, P2, P10), through encouraged discussions on GitHub pull requests or issues (P4, P6, P9), in emails (P2), or in the online chat groups of the development team (P6). Several participants reported to not have publicly documented contribution guidelines (P1, P8, P10, P11, P13), e.g., because a project is closed source (P10), or has only very few contributions (P8). P9 explains that they intentionally avoid being overly strict during code review to maintain a more welcoming experience for first-time contributors [37].

C. Reflection on Applied QA Practices

When we asked the participants how they evaluated their state of quality assurance practices in their projects, we only got concrete answers to the testing aspect. Only for testing they provided concrete answers. Six participant use coverage metrics to judge the adequacy of their test suites (P1, P2, P5, P8, P10, P13). Other ways to judge tests were *intuition* and *exploratory manual testing* (P13), the reviewer's opinion (P4, P5, P10), or whether users report bugs (P9).

For automated workflows, code reviews, and contribution guidelines we could not draw out any comments from our participants judging the adequacy of these practices. Possibly, this is connected to the lack of available developer resources.

D. Motives to Extend QA Practices

While it was difficult to elicit how the engineers reflect on the adequacy of their quality assurance approaches, they more easily shared their motives to extend these practices. This could be seen as an early, pragmatic way of reflection, focused directly on concrete improvements.

The software engineers shared various motives to add new or more tests namely: based on new issues, failures, bugs, or user/customer reports (P3, P4, P5, P6, P7, P8, P11, P13, P14); new features, functionalities, or changes (P2, P3, P4, P7, P10); criticality or complexity of a component (P1, P2, P5, P12); missing coverage (P5, P10); and when higher-level failed to identify the root cause of a failure (P3, P5).

Three participants mentioned rarely updating or adding new automatic continuous integration workflows in their projects (P4, P5, P9). Instead, they extend the components executed by the workflows, e.g., by adding new tests to the test suite. When updates to automated workflows occur, they typically stem from new needs, driven by changes in the project's technology (P3, P4, P6, P9, P13). Having new issues or failures may prompt changes in workflows to help in pinpointing the root cause of problems (P3, P6).

Updates to the automated workflows are motivated by a need for specific information (P3) or personal inspiration from other open source projects (P1).

Three participants gave motives for updating their contribution guidelines such as user reports about problems (P6) or outdated information (P1), or when the project changes (P3).

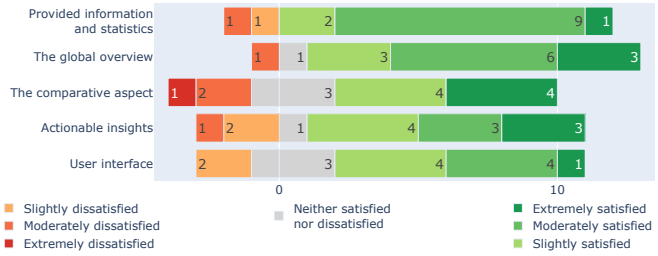
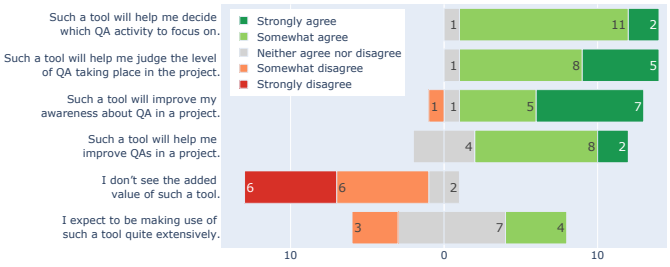
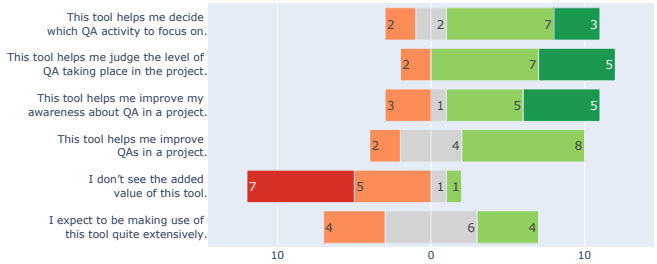


Fig. 4: Participants' satisfaction levels with RepoInsights.



(a) Expectation of a quality analytics dashboard.



(b) Perception of RepoInsights.

Fig. 5: Participants' expectation versus perception.

RQ1. Summary In general participants are positive about QA and know about SA but only a few actively use SA. There is a well-justified variety in the concrete QA practices that projects follow. The concrete usage of QA practices is motivated by the project's goal, context and available resources. A reflected judgment on the adequacy of current QA practices is not common, except testing. Extending QA practices is mainly based on concrete needs.

V. RQ2: What Are Software Engineers' Perspectives on a Quality Assurance Analytics Dashboard Like RepoInsights?

To gauge our participants' perspectives on a quality analytics dashboard, we report their expectations and actual perceptions of `RepoInsights` recorded in the pretest-posttest design. We also characterize the situations in which participants said they would use a QAA dashboard. In Sections V-B and V-C we delve into the qualitative analysis of the interviews to provide a deeper understanding into how `RepoInsights` enabled software engineers reflect on QA and how it could be improved.

Looking at Figure 4, we see that the participants were generally satisfied with the different aspects of `RepoInsights`. The comparative aspect, received the worst ratings, which aligns with our observation that it was also used the least during the assignment.

Regarding the participants' expectations of a QAA dashboard based on our provided description, we see in Figure 5a that they are quite positive about the core intentions of the tool. These encompass helping to decide which QA activities need more focus, judging the level of QA, and improving (the awareness about) QA in a project. The participants see added value in the tool, but are divided on whether they would extensively use it. When we compare this to the perception of the tool after the assignment, we see in Figure 5b that the participants' summarized attitudes did not change much. They lean slightly more critical than before the assignment. This matches with our observations that the participants started to reflect very concretely on the dashboard and its value, opening up many concrete wishes for extensions of the originally described idea of a QAA dashboard.

A. Usage

Overall, all participants were positive about using a dashboard like `RepoInsights` in the future. In terms of frequency of using a QAA dashboard, three participants indicated that they would use it periodically when they are already reflecting on the health of their project (P4, P9, P11). Others mentioned one-off use cases, such as checking the QA status of their project once (P13, P1). As P2 indicates: *"[the tool] helps as a starting point to look into [QA practices]."* P13 points out that the frequency in which new information is available on the dashboard influences how often they would use it.

The role of the software engineer and their habits also play a role here. P14 indicates they are not a person who *"usually looks at dashboards"* but would use it if they were a QA engineer. P6 indicates that it would need to become a habit first, and P14 expect that the main maintainer of a project would use `RepoInsights` more. Two participants feel already quite aware of the state of QA in their projects (P7, P9), which limits additional value.

The participants mentioned further purposes they would use the dashboard for, beyond assessing the QA of their project. P13 suggests to use the QAA dashboard to judge the processes of open source projects they consider as dependencies, P4 proposes troubleshooting release issues such as adapting to a major platform change. P2 mentions inspecting other projects to evaluate their responsiveness to contributions, or to get inspired by their QA tools and workflows to use in their own projects.

B. ***RQ2.1.)*** *In What Ways Did the QAA Dashboard Help Software Engineers Reflect on QA Practices?*

We address this research question by observing how participants interact with the dashboard while completing the assignment. We identify the specific aspects of the dashboard that supported them in reflecting on QA practices.

*1) **Displayed metrics**:* the participants mainly used the dashboard's global overview provided metrics to reflect on QA practices and make judgments about them.

For rating the level of code reviews, seven participants pointed to the average number of discussion comments in pull requests (PRs) (P1, P2, P4, P5, P9, P10, P12). Getting more comments was referred to as positive sign for code reviews. On the other hand, not receiving comments or having few comments was a sign of not having reviews or just a quick code review *"P02: ...no comments in the PR that I would assume the code review was not done or it was not it was just a quick code review..."*. Also, seven participants used the provided review time of PRs, which is the average time it takes to review and merge PRs after their creation (P2, P5, P6, P10, P11, P12, P14). Similarly, six participants referred to the reported average number of participants in PRs (P2, P3, P9, P10, P12, P13), for instance: *"P13: my reasoning is that there are a lot of contributors per single pull request. There's the quality of reviews."*. Six participants took the displayed average change size of PRs into account while making their judgments (P1, P3, P5, P7, P10, P13). This change size includes number of added and deleted lines per PR, and number of changed files. Participants expressed judgments such as the difficulty of reviewing PRs with big changes and how the size of changes provides insight into the stage of development in a project, *"P03: we are seeing a lot of additions and deletions at the same time. That means the project is very much in an exploratory stage or it is in active development"*. Moreover, having a high number of additions and deletions was suspected to be fake and misleading, based on previous experience with GitHub's diff, or also suspected to be from automatic changes, e.g., auto-fixing the code style.

Regarding reflection on the automated workflows (AWs), five participants referred to the number of checks (P1, P2, P3, P4, P11). When the average number of checks is high, participants anticipated thorough testing (P11), overloading and confusing contributors (P3), and increasing difficulty in identifying failing checks (P3). Moreover, P3 viewed an excess of checks as a sign of an exploratory project and pointed out that projects should not have multiple checks for the same purpose. Other metrics used in reflecting on AWs were the reported AWs success rate and the number of failures (P8, P9, P12). The software engineers were generally optimistic regarding an average failure rate of 30% in both projects of the assignment.

When reflecting on testing, five participants based their evaluation merely on the provided code coverage percentage (P2, P3, P7, P10, P13). They considered 85% to be a high percentage for the test suites.

*2) **Meaningful details**:* additional details shown in the dashboard allowed participants to gain a deeper understanding of the context of the projects, enabling them to make more informed judgments.

Regarding judging code reviews, four participants referred to the detailed information on the PR details page (P7, P9, P10, P13). P13 specifically noted the presence of Dependabot-created PRs in one project, which could skew the data. P7 found the outlier PRs appearing on the PR trend graphs to be another point of interest, which have touched thousands of lines while other PRs have mostly changed a few lines.

When judging automated workflows, five participants noted the names of the top failed checks in AWs overview of the dashboard (P4, P7, P8, P9, P13). This detail helped them in gaining a better understanding of the purpose of AWs in projects, which they could then relate to other information such as average number of checks and their runtime. *"P09: [while judging AWs, based on check names] ... they're clearly testing on three platforms."*. Another example of judging details of AW failures was when P1 compared test failures and build failures, expressing skepticism regarding build failures, while test failures were deemed more acceptable, as contributors might not run all tests locally.

Regarding assessing tests, three participants made specific judgments using the code coverage trend graph (P9, P10, P13). They noted that a decreasing trend in code coverage in an active project could indicate concerns about code quality.

*3) **Providing the global overview**:* when reflecting on various QA practices, participants primarily took into account multiple metrics and details related to the projects' QAs. While this might seem straightforward when focusing on individual practices and metrics, our emphasis lies in considering information about multiple QA practices' metrics. The global overview highlights the complementary nature of various QA practices: *"P05: ... So I think a tool that integrated so many statistics and it really helps."*; *"P02: Having everything in one page, like the code review guidelines, I think that's nice"*; *"P01: I really like the fact that you have an overview of possibly all the different things, I like that it's combined."*

*4) **Providing visible links**:* when reflecting on the guidelines of projects, five participants relied solely on the indication of their existence (P2, P3, P7, P8, P12). However, P7 and P8 expressed interest in seeing the content of the guidelines to make more accurate assessments.

We had included links to projects' available issue and PR templates, as well as their code of conduct and contribution guidelines. Initially, at the start of the assignment, we encouraged participants to base their judgments about projects' QA solely on the information provided on the dashboard. However, despite this guidance, P7, P10, and P13 clicked on the links to inspect the content of these files and evaluate their quality.

For a global overview of the project, consolidating all information in one place, particularly raw, unprocessed data, may not be necessary. Instead, incorporating visible links allows access to relevant information, facilitating a thorough understanding of project details and context without overwhelming users with excessive data (e.g., content of guidelines, configurations details, build logs, etc.).

RQ2.1. Summary Participants reflected on projects' QA practices using: the displayed metrics regarding projects' QA practices, the provided meaningful details to get a deeper understanding of projects' context, the global overview aspect, and the visible links to plain data.

C. *RQ2.2.) In What Ways Could the QAA Dashboard Be Improved to Help Engineers Reflect on QA Practices?*

We collected and qualitatively analyzed comments, suggestions, and feedback about `RepoInsights` provided by participants during the interviews. In this section we present the result of this analysis as a set of improvement points.

*1) **Providing high granularity of information**:* eight participants suggested increasing the granularity of information for various QA metrics and statistics (P1, P3, P4, P7, P10, P12, P13, P14). Specifically, they highlighted the necessity of higher granularity to identify areas for improvement and make informed decisions about where to focus efforts. *"P01: ... you need this granularity to really see where you're lacking."*; *"P03: information is very aggregated. I would expect it to be somewhat detailed,"*.

Regarding tests, five participants highlighted the need for detailed information to properly reflect on testing (P1, P7, P10, P12, P13). They expressed that the provided code coverage and its trend alone were insufficient. They recommended incorporating higher granularity for the provided information, including detailed granular coverage of different submodules or subsystems, highlighting parts with missing coverage, and the tests contributing to the coverage of each module.

When reflecting on code reviews, five participants suggested more details, including: acceptance rate of PRs, fine-tuned change information rather than merely additions and deletions per PR, distinguishing different types of comments in PRs and providing a distribution of where PRs touch the code (P3, P4, P7, P13, P14).

In discussions about automated workflows, P1 suggested providing insights into the use of automated workflows to indicate if checks are mandated during the development process. Moreover, P14 suggested adding the date of the last update of automated checks and providing recommendations for updating them in response to major changes in a project. Additionally, P13 and P14 recommended higher granularity in AWs by providing detailed breakdowns of their purpose.

Regarding guidelines, two participants suggested providing more details: P7 suggested analyzing the quality of guidelines through keyword matching or using AI-based analysis to assess their completeness. Additionally, P13 proposed analyzing the differences between issue or PR templates and what users submit, particularly for open-source projects that receive numerous issues and PRs.

*2) **Providing additional data insights**:* to have a deeper understanding of what is happening in the projects, participants suggested incorporating new data insights and statistics. While these suggestions may not directly relate to project QA, they are vital for contextualizing projects and facilitating more insightful reflection: **Additional data regarding tests**: providing more data insights such as adding new metrics for test coverage such as mutation score, number of tests, proportion of commits changing project's tests, lines of added code versus tests, and number of executed/failed tests (P4, P8, P10, P13). **Trends analysis**: incorporating trend analysis to visually depict the evolution of metrics over time. While this feature was provided for certain metrics, such as PR activities and code coverage, participants suggested extending it to include other relevant metrics (P7, P10, P13, P14). **Projects' age and goals**: are related to the availability of guidelines (P1, P3, P7, P9, P11). **More charts and metrics**: offering statistical charts and median values for presented metrics to mitigate the skewness caused by outliers (P1, P3, P13). **Number of contributors**: the intensity of QA practices in projects can depend on the number of contributors (P3, P4, P13). **Development processes**: a long review time can be acceptable if multiple sets of time-consuming checks need to be completed, and that having multiple workflows does not necessarily mean that they are used and enforced in the PRs; so depending on the development process reflection on the QA can vary (P1, P3). **Metrics definition**: adding extra information about the definition of all provided metrics to avoid confusion and enable a more accurate reflection (P4, P9). **Project Releases**: incorporating information about project releases, including the usage of automatic releases, to assess project activity levels and indicate robust QA practices (P13). **Project Activity Overview**: offering an overview of recent project activities to show its current level of engagement (P13). **Issues Statistics**: introducing statistics for issues similar to PRs, such as their waiting duration and topic, to provide a comprehensive view of project maintenance (P4).

*3) **Providing meaningful comparison**:* participants recommended enhancing the comparative aspect by: **Custom comparison**: three participants suggested adding the feature to select specific projects or groups of projects for comparison (P1, P5, P10). **Comparison against similar projects**: two participants (P1, P9) suggested having clusters of similar projects based on their attributes, such as programming language. Another participant suggested the ability to compare against projects within the same GitHub organization.

*4) **Providing the context of projects**:* six participants expressed concern about their lack of familiarity with the context of projects while working on the assignment (P1, P3, P9, P11, P13, P14). *"P13: But probably the developers of those projects will have a different perspective because they know what's happening in their projects."* This context encompasses various factors such as the type of software project, its requirements and goals. Additionally, it involves considerations like available developer resources. Furthermore, more details about the configuration of automated workflows, and the acceptance of outside contributions are (e.g., OSS) or are not (e.g., companies) important. Understanding the context is also crucial when making comparisons with other projects. It is more logical to compare similar projects to gather insights about improving QA practices.

One explanation for this emphasis of our participants on

providing the project context is that they were not familiar with the projects we asked them to judge. Even if users are working on projects they know well, providing sufficient context can enhance their understanding. Beyond that, users may have varying levels of familiarity with different aspects of their projects—precisely what `RepoInsights` tries to address regarding QA—and providing contextual information can help bridge gaps in their knowledge.

Regarding reflection on guidelines, four participants provided specific judgements when no guidelines were available (P3, P4, P5, P15). For instance, they deemed it acceptable if it was a single-developer project, interpreting the absence of guidelines as an indication of only one developer. They also inferred that potential contributors would not know how to contribute to the project without guidelines.

5) ***Providing actionable insights****:* nine participants gave feedback regarding the need to make insights of the dashboard more actionable (P1, P2, P4, P5, P9, P10, P12, P14). More specifically, four participants (P1, P3, P5, P12) pointed to the need of more detailed concrete suggestions to improve actionability of dashboards insights: *"P12: That's what I'm doing is like, you know, looking at these statistics, it might tell you that you need to take some action, but it's not clear what action is required."*. Moreover, they mentioned that insights regarding some QA practices are more actionable, *"P12: I can see the code coverage and that's something that I may need to implement some action on. I can see whether or not there are actual contributor guidelines. That's something that can be actioned on."*, there are other areas that need to be more actionable by suggesting how to improve QAs, for example P12 who mentions that a low number of reviewers during code review is useful to know, but does not provide a solution.

6) ***Providing the ability to filter (out) information****:* four participants recommended adding filtering capabilities to include relevant information and exclude irrelevant data. (P7, P8, P9, P14). The dashboard includes various QA-related data to provide a global overview, however, depending on the project's context, and on the user's preferences, it would be useful to filter out unwanted information. Examples are: Dependabot or any other bot-created PRs that might skew the data, or unimportant build failures that makes it difficult to see what really matters.

Not all projects are interested in all information equally. While the default dashboard view intentionally shows a variety of QA practices to inspire adopting them, users should be able to hide information they do not find necessary for the future times they consult the dashboard. There exists a wide range of approaches to QA practices across projects. Certain projects prioritize extensive testing of complex software components during development (P1), while others solely rely on end-user tests and reported issues for post-production testing (P14). Additionally, code coverage metrics hold high importance for some projects (P1, P3, P8), while in others, developers lack awareness of this concept (P6). These differences underscore the need for a customizable QAA dashboard to accommodate diverse project goals and perspectives.

RQ2.2. Summary Based on participants suggestions and feedback, providing the followings can help them in better reflection: high granularity of information, meaningful comparison, context of projects, actionable insights, and ability to filter (out) information.

Concrete feedback and suggestions: We also received a plethora of feedback and suggestions related to concrete enhancements of the dashboard's user interface and user experience, such as explanatory texts on how the metrics are calculated and what data they precisely include. We documented these points to refine our prototype in future iterations. As this information is mainly specific to the implementation of our prototype, it falls outside the scope of this paper. However, toolmakers and researchers keen on developing similar software analytics tools can access this information through our provided replication package [33].

D. ***RQ2. Summary*** *Summarizing Software Engineers' Perspectives on* `RepoInsights`

The participants were generally satisfied with `RepoInsights`, particularly its global overview, provided information, and statistics. Their perception of `RepoInsights` closely alined with their positive expectations of a QAA dashboard. Most participants expressed interest in future use. From their comments, we formulated four insights on what was successful in the design of `RepoInsights` as well as six suggestions to improve it.

VI. Related Work

Quality assurance practices in software engineering have been studied both individually and in combination [27]. Studies on complementarity of quality assurance practices, such as software testing [2], [4]–[6], modern code reviews [9], automated workflows [24], build automation [16]–[18], and automated static analysis [11], have shown that QA techniques are mostly *not* used in conjunction [27] and software engineers' **situational awareness** regarding them is lacking [21]. In this study, we aimed to overcome this awareness gap [23] with a tailored software analytics dashboard [22].

For more than a decade [38], **software analytics** has played a significant role in enabling software engineering individuals to make informed, data-driven decisions across various facets of software development. Applications include quality assessment tools such as SonarCube[7], Kiuwan[8], and Bitergia[9], as well as those tailored for DevOps environments like New Relic[10], and Datadog[11]. Moreover, research endeavors have contributed to this domain, yielding open-source tools such as GrimoireLab [39], a comprehensive toolset for software development analytics, SQuAVisiT [40], which offers flexible

[7]https://www.sonarqube.com/, last visit: April 11, 2024
[8]https://www.kiuwan.com/, last visit: April 11, 2024
[9]https://bitergia.com/, last visit: April 11, 2024
[10]https://newrelic.com/, last visit: April 11, 2024
[11]https://www.datadoghq.com/, last visit: April 11, 2024

visual software analytics capabilities, Q-Rapids [41], facilitating continuous quality assessment throughout the software development lifecycle, QaSD [42], a quality-aware dashboard designed to enhance software quality and development processes, and QConnect [43], a quality-aware dashboard focused on developers' activity and productivity.

The highlighted tools above, underscore the multifaceted impact of software analytics on software engineering. However, none of them target QA practices. Our goal is to fill this gap to potentially enhance the situational awareness [20] surrounding QA practices and empower software engineers to make informed decisions regarding QA of their projects.

VII. Threats to Validity

Reliability: to ensure consistency in our qualitative analysis, two authors coded half of the interviews independently and met repeatedly to discuss emerging codes until reaching a negotiated agreement [36]. However, other researchers' interpretations of the data may differ. We publish our full codebook to enable validation of our findings [33].

Respondent Bias: participants may have provided socially desirable answers when evaluating a tool developed by the interviewer. To mitigate this, we encouraged constructive criticism and maintained a welcoming attitude toward critical feedback. The wide range of improvement suggestions received suggests this threat was reasonably addressed.

Internal Validity: the open-ended interview format threatens reproducibility. Many key insights emerged from only a few interviews, so repeating the study may miss some perspectives.

External Validity: findings may not generalize beyond the specific QA practices, project context and participants we studied. We selected active and popular projects with ample data on PRs and AWs for the dashboard. We only included projects with available test coverage information. Additionally, we varied code review activity and contributor counts to enable comparisons and observe participants' reactions to different quality assurance levels. We openly recruited participants from industry, academia, and OSS, then, directly recruited two more from industry and academia to ensure diversity.

VIII. Implications and Future Work

Implications. While software engineers are generally positive about quality assurance (QA) and software analytics, our study revealed that reflection on the adequacy of QA practices is not common, except for testing activities (RQ1). From our sample of participants, and from prior work of Khatami and Zaidman, we know that there is currently no widespread adoption of analytics tools to specifically support reflection on QA practices across projects [21]. This represents an opportunity for toolmakers to increase software engineers' situational awareness surrounding QA practices and allow them to reflect holistically on their projects' QA activities.

Our evaluation of this idea, through a prototype of a quality assurance analytics (QAA) tool `RepoInsights` and the subsequent analysis of developers' perspectives, highlighted several key implications: **(1)** there is interest and perceived value in centralized quality analytics that provide a global overview of QA metrics and practices across projects (RQ2.1). Participants were generally satisfied with `RepoInsights`'s ability to surface this higher-level perspective. However, **(2)** QAA tools must go beyond simply displaying metrics. Software engineers prefer high granularity, contextualized information tailored to each project's specific practices and needs (RQ2.2). Furthermore, actionable insights derived from data and presented to the user are essential. **(3)** Successful QAA designs should provide meaningful comparisons between projects, the ability to customize viewed information, and rich context about each project's QA activities (RQ2.2).

Overall, we hypothesize based on the insights of our study, that QAA tools presenting contextualized, actionable insights can empower data-driven evolution of QA strategies as project situations change. Seamless integration into developer workflows is key for increasing adoption and reflection.

Future work. Key areas for future research to maximize the adoption and impact of QAA tools include: adopting and evaluating intuitive visualization techniques to derive tailored, actionable insights from quality data beyond just presenting raw metrics. Investigating integration pathways to embed QAA into developer workflows and toolchains. Evaluating effectiveness and validating QAA tools' by measuring impact on quality and productivity.

IX. Conclusion

In this study, we set out to raise the situational awareness of software engineers surrounding quality assurance (QA) in their projects. Through a pretest-posttest study combined with interviews, we evaluated our conceptual design of a software quality assurance analytics (QAA) dashboard. We elicited the current approaches in adoption and reflection of QA practices from our 14 participants. We observed how our `RepoInsights` dashboard supported them, and we identified a set of recommendations for a dashboard like `RepoInsights` to empower software engineers to better reflect on QA. Our key insights are: reflection is uncommon, except for testing, and extending QA practices is based on concrete needs (RQ1). Participants valued the global overview of metrics/practices and the meaningful details that the QAA dashboard provided (RQ2.1). Desired features included high granularity information, meaningful comparisons, tailored insights, filtering capabilities, and more details to have a deeper understanding of projects' context (RQ2.2).

Overall, our results demonstrate interest in QAA tools that go beyond raw data & metrics to deliver highly granular, contextualized, and actionable insights. We hypothesize that such a QAA tool that takes our recommendations into account, can potentially raise situational awareness and facilitate reflection and improvement of QA practices.

Acknowledgment

This research was partially funded by the Dutch science foundation NWO's Vici grant "TestShift" (VI.C.182.032). We thank all interviewees for their time and valuable insights.

REFERENCES

[1] M. Jazayeri, "The education of a software engineer," in *Proc. International Conference on Automated Software Engineering (ASE)*. IEEE, 2004.

[2] M. Aniche, *Effective Software Testing: A Developer's Guide*. Manning Publications, 2022.

[3] A. J. Ko, B. Dosono, and N. Duriseti, "Thirty years of software problems in the news," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. ACM, 2014, pp. 32–39.

[4] M. Aniche, C. Treude, and A. Zaidman, "How developers engineer test cases: An observational study," *IEEE Trans. Software Eng.*, vol. 48, no. 12, pp. 4925–4946, 2022.

[5] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Trans. Software Eng.*, vol. 39, no. 6, pp. 757–773, 2013.

[6] M. Beller, G. Gousios, A. Panichella, S. Proksch, S. Amann, and A. Zaidman, "Developer testing in the IDE: patterns, beliefs, and behavior," *IEEE Trans. Software Eng.*, vol. 45, no. 3, pp. 261–284, 2019.

[7] M. Beller, G. Gousios, A. Panichella, and A. Zaidman, "When, how, and why developers (do not) test in their IDEs," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 2015, pp. 179–190.

[8] M. Beller, G. Gousios, and A. Zaidman, "How (much) do developers test?" in *37th IEEE/ACM International Conference on Software Engineering (ICSE) — Volume 2*. IEEE, 2015, pp. 559–562.

[9] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *35th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2013, pp. 712–721.

[10] M. Beller, A. Bacchelli, A. Zaidman, and E. Jürgens, "Modern code reviews in open-source projects: which problems do they fix?" in *11th Working Conference on Mining Software Repositories (MSR)*. ACM, 2014, pp. 202–211.

[11] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, "Analyzing the state of static analysis: A large-scale evaluation in open source software," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2016, pp. 470–481.

[12] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, "How developers engage with static analysis tools in different contexts," *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1419–1457, 2020.

[13] D. Han, C. Ragkhitwetsagul, J. Krinke, M. Paixao, and G. Rosa, "Does code review really remove coding convention violations?" in *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2020, pp. 43–53.

[14] T. Buckers, C. Cao, M. Doesburg, B. Gong, S. Wang, M. Beller, and A. Zaidman, "UAV: warnings from multiple automated static analysis tools at a glance," in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 472–476.

[15] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, A. Zaidman, and H. C. Gall, "Context is king: The developer perspective on the usage of static analysis tools," in *25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 38–49.

[16] M. Beller, G. Gousios, and A. Zaidman, "Oops, my tests broke the build: an explorative analysis of Travis CI with GitHub," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 356–367.

[17] A. Rahman, A. Partho, D. Meder, and L. Williams, "Which factors influence practitioners' usage of build automation tools?" in *International Workshop on Rapid Continuous Software Engineering (RCoSE)*, 2017, pp. 20–26.

[18] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 2016, pp. 426–437.

[19] A. Khatami, C. Willekens, and A. Zaidman, "Catching smells in the act: A github actions workflow investigation," in *24th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2024.

[20] O. Baysal, R. Holmes, and M. W. Godfrey, "Situational awareness: personalizing issue tracking systems," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. IEEE Computer Society, 2013, pp. 1185–1188.

[21] A. Khatami and A. Zaidman, "Quality assurance awareness in open source software projects on github," in *23rd IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2023, pp. 174–185.

[22] T. Menzies and T. Zimmermann, "Software analytics: So what?" *IEEE Software*, vol. 30, pp. 31—37, 2013.

[23] O. Baysal, R. Holmes, and M. W. Godfrey, "Developer dashboards: The need for qualitative analytics," *IEEE Softw.*, vol. 30, no. 4, pp. 46–52, 2013.

[24] M. Wessel, T. Mens, A. Decan, and P. R. Mazrae, "The github development workflow automation ecosystems," *CoRR*, vol. abs/2305.04772, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2305.04772

[25] C. D. Wickens, "The trade-off of design for routine and unexpected performance: Implications of situation awareness," *Situation awareness analysis and measurement*, pp. 211–225, 2000.

[26] M. R. Endsley, "Toward a theory of situation awareness in dynamic systems," *Human Factors*, vol. 37, no. 1, pp. 32–64, 1995.

[27] A. Khatami and A. Zaidman, "State-of-the-practice in quality assurance in Java-based open source software development," *Software: Practice and Experience*, vol. 54, no. 8, pp. 1408–1446, 2024.

[28] S. Panichella, V. Arnaoudova, M. D. Penta, and G. Antoniol, "Would static analysis tools help developers with code reviews?" in *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 161–170.

[29] Mern stack explained. [Online]. Available: https://www.mongodb.com/mern-stack

[30] D. Campbell, J. Stanley, and N. Gage, *Experimental and quasi-experimental designs for research*. Chicago: Rand McNally, 1963.

[31] E. Babbie, *The practice of social research*, 11th ed. Belmont: Wadsworth, 2007.

[32] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for MSR studies," in *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*. IEEE, 2021, pp. 560–564.

[33] A. Khatami, C. Brandt, and A. Zaidman, "Replication Package for "Software Quality Assurance Analytics: Enabling Software Engineers to Reflect on QA Practices" Paper (SCAM 2024)," Apr. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.10961021

[34] A. L. Strauss and J. M. Corbin, "Basics of qualitative research: Techniques and procedures for developing grounded theory," *SAGE Publications*, 1998.

[35] B. G. Glaser and A. L. Strauss, *Discovery of Grounded Theory: Strategies for Qualitative Research*. Routledge, 2017.

[36] D. R. Garrison, M. Cleveland-Innes, M. Koole, and J. Kappelman, "Revisiting methodological issues in transcript analysis: Negotiated coding and reliability," *The internet and higher education*, vol. 9, no. 1, pp. 1–8, 2006.

[37] G. Gousios, A. Zaidman, M. D. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *37th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2015, pp. 358–368.

[38] T. M. Abdellatif, L. F. Capretz, and D. Ho, "Software analytics to software practice: A systematic literature review," in *1st IEEE/ACM International Workshop on Big Data Software Engineering (BIGDSE)*. IEEE Computer Society, 2015, pp. 30–36.

[39] S. Dueñas, V. Cosentino, J. M. González-Barahona, A. del Castillo San Felix, D. Izquierdo-Cortazar, L. Cañas-Díaz, and A. P. García-Plaza, "Grimoirelab: A toolset for software development analytics," *PeerJ Comput. Sci.*, vol. 7, p. e601, 2021. [Online]. Available: https://doi.org/10.7717/peerj-cs.601

[40] M. van den Brand, S. A. Roubtsov, and A. Serebrenik, "Squavisit: A flexible tool for visual software analytics," in *13th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE Computer Society, 2009, pp. 331–332.

[41] S. Martínez-Fernández, A. M. Vollmer, A. Jedlitschka, X. Franch, L. López, P. Ram, P. Rodríguez, S. Aaramaa, A. Bagnato, M. Choras, and J. Partanen, "Continuously assessing and improving software quality with software analytics tools: A case study," *IEEE Access*, vol. 7, pp. 68 219–68 239, 2019.

[42] L. López, M. Manzano, C. Gómez, M. Oriol, C. Farré, X. Franch, S. Martínez-Fernández, and A. M. Vollmer, "Qasd: A quality-aware

strategic dashboard for supporting decision makers in agile software development," *Sci. Comput. Program.*, vol. 202, p. 102568, 2021.

[43] H. M. Shah, Q. Z. Syed, B. Shankaranarayanan, I. Palit, A. Singh, K. Raval, K. Savaliya, and T. Sharma, "Mining and fusing productivity metrics with code quality information at scale," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2023, pp. 563–567.