The Journal of Systems & Software 188 (2022) 111259

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration ☆

Yuqing Wang *, Mika V. Mäntylä, Zihao Liu, Jouni Markkula

M3S research unit, University of Oulu, Pentti Kaiteran katu 1, Oulu, 90014, Finland

ARTICLE INFO

Article history:
Received 19 April 2021
Received in revised form 10 January 2022
Accepted 3 February 2022
Available online 11 February 2022

Keywords:
Continuous integration
Test automation
Best practice
Software repository mining
Software testing
Empirical software engineering

ABSTRACT

The popularity of continuous integration (CI) is increasing as a result of market pressure to release product features or updates frequently. The ability of CI to deliver quality at speed depends on reliable test automation. In this paper, we present an empirical study to observe the effect of test automation maturity (assessed by standard best practices in the literature) on product quality, test automation effort, and release cycle in the CI context of open source projects. We run our test automation maturity survey and got responses from 37 open source java projects. We also mined software repositories of the same projects. The main results of regression analysis reveal that, higher levels of test automation maturity are positively associated with higher product quality (p-value=0.000624) and shorter release cycle (p-value=0.01891); There is no statistically significant evidence of increased test automation effort due to higher levels of test automation maturity and product quality. Thus, we conclude that, a potential benefit of improving test automation maturity (using standard best practices) is product quality improvement and release cycle acceleration in the CI context of open source projects. We encourage future research to extend our findings by adding more datasets with different programming languages and CI tools, closed source projects, and large-scale industrial projects. Our recommendation to practitioners (in the similar CI context) is to utilize standard best practices to improve test automation maturity.

1. Introduction

Changing customer requirements and emerging technologies are driving the frequent changes to software products (Fitzgerald and Stol, 2017; Karvonen et al., 2017). Software organizations must rapidly release product features or updates on a continuous cycle (Fitzgerald and Stol, 2017). Continuous Integration (CI) is widely adopted to enable rapid and frequent releases (Ståhl and Bosch, 2014). CI is a software development practice that requires developers of the team to frequently integrate code changes (Fowler and Foemmel, 2006). According to the official website of Travis CI[1] (a popular CI server tool), *"Over 900k open source projects and 600k users are testing on Travis CI"*. Software projects and users on other popular CI server tools like Jenkins and CircleCI also accumulate to a considerable amount. Based on the statistic (MarketsandMarkets Research, 2018), *"CI global market size was USD 402.8 million in 2017 and is expected to reach USD 1,139.3 million by 2023, at a compound annual growth rate of 18.7%"*.

Test automation has a history of over three decades, since around 1990 (Garousi and Elberzhager, 2017). After CI was introduced in 2006, test automation became the backbone of CI (Ståhl and Bosch, 2014; Fowler and Foemmel, 2006). CI requires that each integration of code changes must be verified by a build that automatically executes tests to detect defects early (Fowler and Foemmel, 2006). However, based on many sources (e.g., Ghaleb et al., 2019; Rausch et al., 2017; Ståhl et al., 2017; Hilton et al., 2016), in the CI context, immature test automation practices can lead to negative outcomes, e.g., ineffective in defecting integration effects, cost and schedule overruns, and slow feedback loop, thus, product quality suffers and releases delay — the failure of CI. Maturing test automation practices is essential for the success of CI in the current software industry (Shahin et al., 2017; Ståhl et al., 2017). The software industry and research community refer to immature test automation practices as a lack of test automation maturity (Fewster and Graham, 1999; Wang et al., 2020a; Capgemini et al., 2020; Wang et al., 2022). Prior scholars described that, at the high level of test automation maturity, *"test automation practices are defined, managed, measured, controlled, and effective within the organization"* (Eldh et al., 2014; Wang et al., 2020a; Garousi and Elberzhager, 2017;

☆ Editor: Raffaela Mirandola.

* Corresponding author.
E-mail addresses: yuqing.wang@oulu.fi (Y. Wang), mika.mantyla@oulu.fi (M.V. Mäntylä), zihao.liu@student.oulu.fi (Z. Liu), jouni.markkula@oulu.fi (J. Markkula).

[1] https://travis-ci.org/.

https://doi.org/10.1016/j.jss.2022.111259

Table 1
Inclusion criteria.

C1.	The study is written in English.
C2.	The study is full-text accessible.
C3.	The study is not a duplication of others.
C4.	The study is published in journals, conferences, workshops.
C5.	The study presents findings to depict the relationships between test automation maturity, product quality, test automation effort, and release time in software development.

Fewster and Graham, 1999; Wang et al., 2022). The existing literature has presented a set of standard best practices to guide organizations to reach the high level of test automation maturity (Fewster and Graham, 1999; Wang et al., 2019; Wang, 2018; Wang et al., 2022). For instance, in many test maturity models such as TMap (Vroon et al., 2013), TestSPICE 3.0 (TestSPICE SIG, 2014), and TAIM (Eldh, 2020), an example best practice is — define a test automation strategy to conduct test automation activities under given boundary conditions.

However, an essential issue is whether standard test automation best practices (in the literature) would fit the CI context and enable CI success. Essentially, successful CI practices can deliver high quality products fast on a continuous cycle with reasonable costs. That is, to enable CI success, by using standard best practices in the literature, high product quality should be achieved without the expense of long release time and high costs. Yet, research on that is limited, as CI success was actively researched from CI process related factors (e.g., integration frequency, integration serialization and batching, building status communication (Ståhl and Bosch, 2014; Ghaleb et al., 2019; Ståhl et al., 2017) but few from a test automation maturity based view. As such, many CI practitioners are staying tuned and waiting for further evaluation before they actually use standard best practices to assess and improve their test automation practices (Capgemini et al., 2020; Wendler, 2012; Zampetti et al., 2020; PractiTest, 2020).

In this paper, we present an empirical study to observe the effect of test automation maturity (assessed by standard best practices in the literature) on product quality, test automation effort, and release cycle in the CI context of open source projects. The CI context there refers to the context in where test automation is adopted with CI tools. Our study observed test automation effort as the combination of test automation development effort and execution effort. The reason is, based on practitioner surveys (Hilton et al., 2016; Krill, 2013), approximately 60% of CI costs are spent on test automation development and execution, while these two types of effort together influence the release time in CI practices. Our research question is:

- **Research question:** Do higher levels of test automation maturity lead to better product quality without increased test automation effort and release time in the CI context of open source projects?

To answer our research question, we built a conceptual model, using literature, to hypothesize the relationships among our observed variables: Test automation maturity, Product quality, Test automation effort, and Release cycle. From over 30k open source java projects, we selected 149 ones and sent out the Test automation maturity survey to their main contributors to explore the state of practice of test automation maturity in their CI context. We studied 37 projects, which have main contributors answered our survey, by analyzing survey responses and mining their repositories (GitHub repositories, CI repositories, issue tracker systems) to get metric data. Our dataset contains test automation maturity survey responses, project data, test suite, and test logs of these 37 projects. That is made publicly available (https://doi.org/10.5281/zenodo.5831609). Using the quantitative evidence, we made the following observations:

- Higher levels of test automation maturity are highly significantly associated with higher product quality. The effect of test automation maturity is more significant than product size, product complexity, product popularity, product age, team size, and integration frequency on product quality. Older products exhibit lower product quality.
- Increased test automation effort caused by improving the level of test automation maturity and product quality is not evidenced. The effect of product complexity and team size is more significant than test automation maturity and product quality on test automation effort. More complex products spent less test automation effort. Larger teams spend more test automation effort.
- Higher levels of test automation maturity are significantly associated with shorter release cycles. Test automation effort does not seem to impact release time. The direct effect of product quality on release time is negative.

Our observations show that, in the CI context of observed open source projects, a potential benefit of improving the level of test automation maturity (using standard best practices in the literature) is product quality improvement and release cycle acceleration, while increased test automation effort caused by improving the level of test automation maturity and product quality is not evidenced. This suggests that using standard best practices in the literature to improve test automation maturity can enable CI success of open source projects.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 describes our research method. Section 4 reports study results. Section 5 discusses study findings, explores the implications for researchers and practitioners, and states the limitations. Section 6 concludes the paper.

2. Related work

A literature review was done to search for related work. That follows some guidelines of "Systematic literature studies using snowballing" from Wohlin (2014) but not all. In particular, the guidelines that were followed are the usage of search strings, dedicated search places, explicit inclusion criteria, and documented process. To search for related work, we applied search strings "software AND test* AND automat* AND (maturity OR mature OR improv*) AND quality AND (release OR effort)" and "software AND test* AND automat* AND (maturity OR mature OR improv*) AND continuous AND integration" in Google Scholar. Though each string got above thousands of search results in Google Scholar, the relevant studies were only presented in the first few pages. The light reading was performed on relevant studies presented in Google Scholar. We did not identify studies working on our research question but found a set of studies (Wang et al., 2020b; Hilton et al., 2017; Khomh et al., 2012; Williams et al., 2009; Kasurinen et al., 2010; Berner et al., 2005) that have observed the relationships between our observed variables (test automation maturity, product quality, test automation effort, and release time) in software development. This set of studies meet inclusion criteria in Table 1. Next, the Forward snowballing approach was used to screen additional studies based on citations of that set of studies. This screened additional 743 studies in total. The first and

third authors read the title, abstract, introduction, and conclusion of each study against inclusion criteria (Table 1). They used 'yes' or 'no' to vote for whether a source meets each inclusion criterion. Only 7 studies got 'yes' for all inclusion criteria from each author were selected. In the end, in total, 13 studies were selected as related work. We read the full text of each study to identify relevant findings that depict the relationships between any of our observed variables. A summary of these studies is in Table 2. Column 'Variables' shows which variables are observed in the study; Column 'Findings' concludes the relevant findings. Next, we provide an overview of these study findings.

As noted in Table 2, in all studies (Wang et al., 2020b; Lin et al., 2020; Kumar and Mishra, 2016; Puri-Jobi, 2015; Hilton et al., 2017; Lee and Hwang, 2012; Collins and de Lucena, 2012; Tosun et al., 2009; Williams et al., 2009; Berner et al., 2005) that have observed how test automation maturity affects product quality, the findings verified the positive impact. These studies view product quality as to which extent a software product free from defects during the quality assurance process or in the production environment. It is different from scholars who defined product quality as conformance to a standard or to which extent a software product bears on its ability to satisfy given needs (Int'l Standards Organization, 2001; García-Mireles et al., 2015). Studies from Wang et al. (2020b), Hilton et al. (2017), Lee and Hwang (2012), Berner et al. (2005), Puri-Jobi (2015) and Tosun et al. (2009) reported that better product quality can be achieved after improving test automation maturity against certain standard best practices.

The relationship between test automation maturity, product quality, and test automation effort has been discussed from two different viewpoints. One school of viewpoint states the negative relationship of test automation maturity and product quality with test automation effort. Studies from Lin et al. (2020), Hilton et al. (2017), Lee and Hwang (2012), Williams et al. (2009), and Kasurinen et al. (2010), which hold this view, have observed the increased effort to develop and execute more rigorous automated tests in attaining higher product quality. An experience study (Ramler et al., 2014) reported that the direct effect of test automation maturity on test automation effort is negative. However, a different school of viewpoint reported by studies from Kumar and Mishra (2016), Puri-Jobi (2015), and Tosun et al. (2009) views that, test automation effort will be reduced after improving the level of test automation maturity and product quality after a certain period of time — even though the considerable effort is required to develop and execute rigorous automated tests in attaining high product quality, the later effort savings may offset the invested effort.

The relationship between test automation maturity, product quality, and release cycle also has been viewed differently in prior studies. One view considers that release time must increase against improvements in the level of test automation maturity and product quality. Studies from Kasurinen et al. (2010) and Williams et al. (2009) respectively validated this view with qualitative evidence (by interviewing test professionals) and quantitative evidence mined from a software project. This view supports the school of viewpoint that states the negative relationship of test automation maturity and product quality with test automation effort, as noted in the above paragraph. That is, developing and executing rigorous automated tests to ensure high product quality requires considerable effort, so that the elapsed time is added for product development and thus product releases are delayed. In contrast, an alternative view is, the high level of test automation maturity and product quality is accompanied by short release cycles. Studies from Wang et al. (2020b), Collins and de Lucena (2012), and Berner et al. (2005) reported this view based upon the authors' experience, while Kumar and Mishra (2016) and Puri-Jobi (2015) validated the view in software projects.

Based on the above, prior study findings are not sufficient to answer our research question for many reasons. First, prior studies have declared different viewpoints for the relationships among test automation maturity, product quality, test automation effort, and release cycle. There is a need to evaluate prior viewpoints and solve the conflicts in these viewpoints in CI context of open source projects. Second, only four studies clearly indicated that they focused on the CI context. Third, most prior studies (10/13) reported the experience of individual practitioners or the single case of software organizations/products/projects. To evaluate the viewpoints and draw an overall picture of the software industry, cross-site quantitative evidence is needed (Seaman, 1999). Last, our research aims to validate whether standard best practices in the literature should be recommended for CI practitioners. Among all identified studies, only studies from Wang et al. (2020b), Lee and Hwang (2012) and Hilton et al. (2017) had the similar aim. Their findings were based on authors' experience. Many standard test automation best practices in the literature were not included, e.g., prioritizing important automated tests for execution, providing enough resources, and using the right test automation metrics to measure test automation performance (Wang et al., 2019). To sum up, our empirical study can complement prior studies and make the novelty from the following aspects:

- Our study is the first attempt to empirically evaluate the relationships between test automation maturity, product quality, test automation effort, and release cycle in the CI context of open source projects with quantitative evidence.
- Our study observed open source projects by surveying their main contributors and mining their repositories (GitHub repositories, CI repositories, and issue tracking systems). Software repository mining methodology was barely used in prior studies. Only studies from Williams et al. (2009) and Khomh et al. (2012) have used similar methodology, however, they only mined data from the repositories of a single team or software product, see Table 2.
- Our study was recently finished and intended to present the current view for this research scope. As shown in Table 2, most of the prior studies were published several years ago, while only Wang et al. (2020b), and Lin et al. (2020) carried out recent studies.
- Our findings evidenced some observations of prior studies, resolved the different viewpoints of prior studies, and identified novel research topics to extend the impact and scope of CI and test automation literature.

3. Research method

Our research process has five stages: Conceptual model building, Project selection, Survey design and execution, Project data collection, Data analysis. Each stage is described below.

3.1. Conceptual model building

To address our research question, we built a conceptual model (Fig. 1) to hypothesize the relationships between test automation maturity, product quality, test automation effort, and release cycle. Harter et al. (2000) have empirically observed the relationships between software process maturity, product quality, development effort, and cycle time in software development and accordingly proposed a conceptual model. As test automation is a type of software process, to build our model, we adapted Harter et al.'s model by mapping "Process maturity" onto "Test automation maturity" and mapping "Development effort" onto "Test automation effort". Then, our model is like Harter et al.'s

Table 2
Related work.

Study	Year	Observation	Methodology	Variables	Findings
Wang et al. (2020b)	2020	A test automation maturity improvement program in a DevOps team from a Finnish software company	Experience report (based on meetings, a collection of experience notes, team reflection reports, and telemetry result reports).	Test automaton maturity; release cycle; product quality	The observed team improved its level of test automation maturity in the CI context using several standard best practices: improve product testability, hire/train expert team members, select and integrate suitable tools, promote communication and collaboration, encourage team members. This team reported that, the higher level of test automation maturity is associated with shorter release cycles and better product quality in the CI context.
Lin et al. (2020)	2020	148 developers of open-source android apps	Practitioner survey	Test automation maturity; product quality; test automation effort	91% of the developers (134/148) confirmed that test automation maturity contributes to app quality, but many reported increased efforts to improve the level of test automation maturity for the promise of high quality apps.
Kumar and Mishra (2016)	2016	Three software systems	Case study (based on quantitative observations on metric data from these three software systems)	Test automation maturity; Product quality; Release cycle; Test automation effort	The observations showed that, in these three software systems, test automation maturity have a positive effect on improving product quality, shorting release time, and reducing test automation effort.
Puri-Jobi (2015)	2015	A test automation project in CI context of a communication system	Case study	Test automation maturity; Product quality; Test automation effort; Release cycle	In the observed project, test automation practices were improved against the industrial standards. The result was the increased product quality and reduced test automation effort and release time.
Hilton et al. (2017)	2015	16 software developers from 14 different organizations	Interviews	Test automation maturity; product quality; test automation effort	In the practical CI context, more mature test automation practices can cause better product quality at a cost of increased effort to design rigorous automated tests and execute them in frequent builds.
Ramler et al. (2014)	2014	A company's programmable logic controller software for machinery	Experience report	Test automation maturity; test automation effort	When testing this software, the overall effort invested in adopting disciplined practices to develop rigorous automated tests (reflected as test automation maturity) was high.

(*continued on next page*)

Table 2 (*continued*).

Study	Year	Observation	Methodology	Variables	Findings
Lee and Hwang (2012)	2012	Test maturity improvement program at a small enterprise for developing an medical information system	Experience report	Test automation maturity; Product quality; Test automation effort	The experience showed that, when test automation practices were improved according to ISO/IEC 29119 (consists of standard practices and techniques), the number of defects after releases (measured for product quality) was largely reduced, though much effort was spent to standardize test automation practices (incl. design, execution, planning).
Collins and de Lucena (2012)	2012	A software project developed in the CI context	Experience report	Test automation maturity; Product quality; Release cycle	The project had 20 sprints. Within the project, defects found per sprint were decreased and release speed was increased in the CI context, after improving the level of test automation maturity.
Khomh et al. (2012)	2012	The software development process of Mozilla Firefox during a period when it transitioned to short release cycles	Case study (based on data mined from project wiki, project repository, crash repository, bug repository).	Release cycle; Product quality	In the case of Mozilla Firefox, there is no significant association between release cycle frequency and product quality.
Tosun et al. (2009)	2009	A software quality improvement project within a health care company in Turkey	Experience report	Test automation maturity; product quality; test automation effort	In the observed project, process maturity was improved against best practices (include some test automation best practices) defined in CMMI (a software process maturity model), and the result was the decrease of 4.5% defect rate and 17% of testing effort.
Williams et al. (2009)	2009	One Microsoft team consisting of 32 developers	Case study based on software repository mining, survey, interview, and action research	Test automation maturity; Product quality; Test automation effort; Release cycle	In the observed team, software product quality improved at a cost of approximately 30% more development time (that delayed release time), when test automation was incrementally performed with disciplined practices under the test-driven development context.

(*continued on next page*)

Table 2 (*continued*).

Study	Year	Observation	Methodology	Variables	Findings
Kasurinen et al. (2010)	2009	55 testing specialists from 31 organizations	Interviews	Test automation maturity; test automation effort; Release cycle	Test automation practices require considerable effort (in terms of planning effort, design effort, execution effort, and test result analysis effort) to be mature. That may delay release time.
Berner et al. (2005)	2005	5 practical test automation projects the authors have participated	Experience report	Test automation maturity; release cycle; product quality	Test automation maturity achieved by using several standard best practices (define a good test automation strategy, carefully design and reuse automated tests, and build a testable SUT) can lead to shorter release cycles and better product quality.

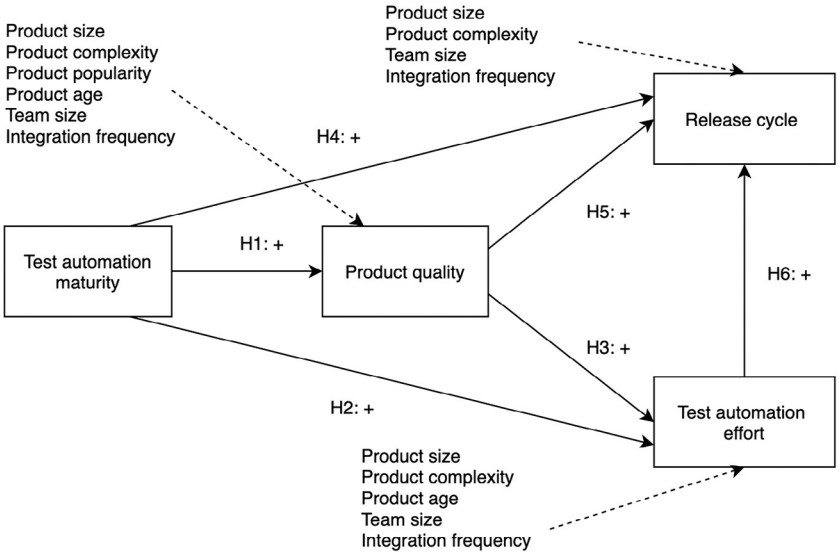


Fig. 1. Conceptual model.

model but with the particular focus on test automation maturity as a type of software process maturity that should have a positive effect on product quality. We also ensured that our model is compatible with previous test automation literature and software engineering literature. The detailed explanation for that is presented in the following sub-sections.

3.1.1. Observed variables

Our conceptual model contains four observed variables. These four variables were derived from our research question and they were conceptualized as below:

- **Test automation maturity:** the level of test automation maturity assessed by standard best practices in the literature. That is, to which extent test automation practices within an organization are close to standard test automation best practices defined in the literature.
- **Test automation effort:** the total effort to develop and execute automated tests.
- **Release cycle:** the length of release cycles — the average time it takes to release a product from the start of development (Harter et al., 2000).
- **Product quality:** our conceptual model utilizes the common definition of product quality as prior studies in Table 2. It considers product quality as to which extent a software product free from defects in the production environment.

3.1.2. Hypothesis

Our conceptual model makes six hypotheses grouped into three aspects:

Effect of test automation maturity on product quality. Harter et al.'s model observed that, high levels of software process maturity (assessed by standard best practices) aid in detecting early defects and reducing defect injection, and thus, are accompanied by high product quality. In test automation literature, prior studies from Wang et al. (2020b), Puri-Jobi (2015), and Hilton et al. (2017) (Table 2) also reported that, better product quality can be achieved using certain standard best practices for test automation maturity improvement in the CI context. This implies the hypothesis:

- **H1 (Test automation maturity and Product quality).** Higher levels of test automation maturity are associated with higher product quality.

Effect of test automation maturity and product quality on test automation effort. Harter et al.'s model observed that, increased development effort was required to follow standard best practices to achieve higher levels of software process maturity and product quality. As discussed in Section 2, in test automation literature, the relationship between test automation maturity, product quality, and test automation effort has been discussed from two different viewpoints. To be compatible with Harter et al.'s observation, our conceptual model follows the view, that implies, there is an inverse relationship of test automation maturity and product quality with test automation effort. Thus, two hypotheses are built:

- **H2 (Test automation maturity and Test automation effort)**. Higher levels of test automation maturity are associated with increased test automation effort.
- **H3 (Product quality and Test automation effort).** Higher product quality is associated with increased test automation effort.

Effect of test automation maturity and product quality on release cycle. Harter et al.'s model observed that, increased development efforts invested in achieving higher levels of software process maturity and product quality would lead to the delay of cycle time. As noted in Section 2, in test automation literature, the relationship between test automation maturity, product quality, and release cycle also has been viewed from two different viewpoints. To be compatible with Harter et al.'s observation, our model follows the view, which indicates cycle time will increase against improvements in the level of test automation maturity and product quality. This supposes:

- **H4 (Test automation maturity and Release cycle).** Higher levels of test automation maturity are associated with longer release cycles.
- **H5 (Product quality and Release cycle).** Higher product quality is associated with longer release cycles.
- **H6 (Test automation effort and Release cycle).** Increased test automation effort is associated with longer release cycles.

3.1.3. Control variables

Scientists use control variables to ensure their experimental results are solely caused by their observations (Field and Hole,

2002). In our conceptual model, we identified control variables that may affect our observations on the relationships among our observed variables. Each control variable is explained below:

- **Product size:** how many lines of code (LOC) a software product contains (Harter et al., 2000).
- **Product complexity:** "software complexity is a natural byproduct of the functional complexity that the code is attempting to enable" (Samli et al., 2020).
- **Product popularity:** how many forks a product has (Vasilescu et al., 2015).
- **Product age:** software product age is the time a software product is created.
- **Team size:** how many people are work in software development (Vasilescu et al., 2015).
- **Integration frequency:** how often the code changes are integrated in CI context (Ståhl et al., 2017).

Product quality. Studies from Banker (1992), Agrawal and Chari (2007) and Zazworka et al. (2011) empirically observed that products in larger size have more defects. Many scholars (Agrawal and Chari, 2007; Zazworka et al., 2011; MacCormack et al., 2006) observed a positive correlation between product complexity and product quality: more complex software products tend to have more defects. Harter et al. (2000) investigated that newer products expose more defects. A recent study (Vasilescu et al., 2015) examined that products that are widely forked may have a high opportunity to expose defects. Many studies (Cotroneo et al., 2016; Ibrahim et al., 2012; Au et al., 2009) reported that software products developed in a larger team size tend to have more defects, because "defects are human being caused" and thus more people means a higher occurrence chance of defects. Studies from Ståhl et al. (2017), and Vasilescu et al. (2015) examined that product quality is affected by integration frequency.

Test automation effort. Based on many studies (Ståhl et al., 2017; Harter et al., 2000; Agrawal and Chari, 2007), product size is an important factor that explains test automation effort, as test scope and frequency tend to increase along with product size growth. Several scholars (Harter et al., 2000; Agrawal and Chari, 2007) empirically observed that it would take more effort to develop and execute automated tests on software products with higher complexity. As noted by the study (Kumar and Mishra, 2016), older products spend less test automation effort as they can reuse the existing automated tests. Moreover, many studies (Ståhl et al., 2017; Banker, 1992; Zhao et al., 2017; Shahin et al., 2017) have noted that larger teams spend more test automation effort in the CI context. More developers in the team indicate an increased rate of changes – to make a software product to be ready on each committed change, test automation should reach the proper scope and deep – thus test automation effort is increased (Ståhl et al., 2017; Banker, 1992; Zhao et al., 2017; Shahin et al., 2017). Prior scholars (Hilton et al., 2016; Hamdan and Alramouni, 2015) validated that integrating code changes cost test automation effort in CI builds.

Release cycle. Studies from Harter et al. (2000) and Agrawal and Chari (2007) found that product size and product complexity are negatively associated with release time. Larger products take longer to be developed and therefore are accompanied by longer release cycles (Harter et al., 2000; Agrawal and Chari, 2007). More complex products are delivered in longer release cycles, because the longer time is required to code more complex functions and methods inside of them (Harter et al., 2000; Agrawal and Chari, 2007). The study (Vasilescu et al., 2015) evidenced that the team size could affect software development productivity that determines the release time in the CI context. Studies from Hamdan and Alramouni (2015), Ståhl et al. (2017) and Pinto et al. (2018) observed that frequent integration would lead to short release cycles.

Table 3
Summary of project selection results.

Dataset	Original	Step 1	Step 2
JTec (Corò et al., 2020)	31222	1459	126
20-MAD (Claes and Mäntylä, 2020)	765	38	23

Table 4
Initial selection criteria.

C1.	Is not a fork repository of the other.
C2.	Use Travis CI and its Travis CI repository is accessible to the public.
C3.	Run automated tests with Maven under Travis CI environment.
C4.	Use Github issue tracker or Jira.

3.2. Project selection

To validate our conceptual model (Fig. 1) in the CI context of open source projects, it was decided to select open source java projects that adopt test automation in the CI context. As the performance and structure of different programming languages are different, to avoid bias of the research, we must select projects written in the same programming language. Java projects were focused since they are popular and exist in large numbers; This would enable us to find enough relevant projects for our study. We began with JTeC (Corò et al., 2020) and 20-MAD (Claes and Mäntylä, 2020) datasets that have collected the amount of open source java projects. The project selection process consists of two steps. Table 3 summarizes each step's result.

In step 1, we screened projects from JTeC and 20-MAD datasets against the initial selection criteria in Table 4. Projects that did not meet all criteria were excluded. C1 ensured that a project is original. C2 checked whether a project adopts CI and its CI repository is accessible to us. We only looked at projects that use Travis CI. Travis CI is a popular CI server tool for open source projects and it allows to access the building history via its API. That would allow us to collect test automation and CI related data for our study. C3 verified whether a project has run automated tests in its CI context, and limited the building tool to Maven. As the performance of different CI building tools to compile and run automated tests is different (Hilton et al., 2016), to avoid bias of the research, it is necessary to focus on one CI building tool. Maven has a standard output for test execution and allows externals to mine testing related data. It is dominant in Apache open source java projects,[2] which exist in a large number on our starting datasets. Ignoring Apache projects would lose the large number of candidate projects. C4 selected the projects that report bugs using Github issue tracker or Jira, which are common issue tracking systems for java projects. At end of step 1, the initial selection criteria resulted in a candidate set of 1,497 projects.

In step 2, we further screened the candidate projects selected from step 1 against project active selection criteria in Table 5. These criteria were used to select projects, which are active to commit changes (AC1), run CI builds (AC2), adopt test automation with our selected CI tool Maven (AC3), report defects in issue tracker system (AC4-6), and publish releases (AC7) at the time of doing our research. At the start, we examined prior software repository mining studies (Hilton et al., 2016; Vasilescu et al., 2015; Rausch et al., 2017; Cataldo and Nambiar, 2009; Subramanian et al., 2007) on our observed variables to set threshold values in these criteria. Many attempts were conducted to test threshold values, e.g., changing from a dozen to thousands of commits, CI builds, automated tests, reported defect-related issues, and the number of releases. The final set of threshold values was chosen according to the results of our attempts, as it allowed us to

[2] https://github.com/apache.

Table 5
Project active selection criteria.

AC1.	Had at least 90 commits in 2020
AC2.	Run at least 10 CI builds under Travis CI environment in 2020
AC3.	Run automated tests with Maven in CI builds in 2020
AC4.	Had more than 90 issues reported in Github Issue Tracker or Jira
AC5.	Used tags to label issues — at least 60% of issues have tags
AC6.	Used naming convention (e.g., bug, defect, fault, error, flaw, and other synonyms) to label defects in Github Issue Tracker or Jira
AC7.	Had at least a release in 2020

select projects (from datasets) that contain enough metric data on our observed variables and ensure sample diversity (Linåker et al., 2015) for valid observations. Our attempts found that, most projects that did not meet AC1 are inactive — their project repositories were barely updated or changed in the current year. Collecting data from such projects may make invalid observations for their current test automation practices. Regarding AC2, our attempts showed that, projects did not meet AC2 and AC3 having few automated tests and these projects exist in large numbers. Including these projects may introduce too many outliers (Miller, 1993). That may have the disproportionate effect on later statistical results. We also examined that, setting the threshold value of CI builds to bigger numbers (e.g., 20, 50, 100) in AC2 are likely to neglect projects, which are in small size or just start CI and test automation. Neglecting such projects may harm the sample diversity. Our attempts showed that, most of the projects that did not meet AC4–AC6 were not active in issue tracker systems — we were only able to recognize 0–2 bugs from the issue tracker system of each of such projects. Including these projects may bias our research, which intended to measure product quality of projects using reported bugs from issue tracker systems. AC7 was set to select projects with varied release cycles but reject the null value for statistics. As a result, we finally selected 149 projects that met all criteria for further study.

3.3. Survey design and execution

To explore the state of practice of test automation maturity (regarding the adoption of standard best practices in the literature) in selected 149 projects, we conducted a test automation maturity survey with the main contributors of each project. For survey design and execution, authors consulted 'Guidelines for Conducting surveys in Software Engineering' from Linåker et al. (2015) and the general survey guidelines from Groves et al. (2011).

Sampling plan. To ensure the accuracy of the sample, the first author created a sampling plan and it was reviewed and revised with other authors. In survey studies, two common sampling methods are widely used: probabilistic sampling and non-probabilistic sampling (Punter et al., 2003; Groves et al., 2011). Probabilistic sampling assumes that every member of the target population is available and there is a random process to select participants (Groves et al., 2011). Non-probabilistic sampling requires selecting the representative sample (Groves et al., 2011).

In our survey, the target population was defined as the main contributors of selected 149 projects. Non-probabilistic sampling method was used for two reasons. First, due to European General Data Protection Regulation (GDPR), it was impossible to collect all individual contributors' contacts and send out the survey to them without their permission. Second, open source projects allow for the involvement of both internal and external contributors. In our survey, it was expected to only select internal contributors, who currently are core members of a project and really understand test automation practices and CI practices in this project. As a consequence, for each of 149 projects, we visited its project page and browsed the list of internal contributions, and then, only selected contributors meeting the following criteria:

- Currently is the core member of the project.
- Had more than 10 commits to the project in recent 6 months.
- Choose to make their email addresses visible to the public. This criterion was defined with respect to GDPR.

Each project had 3–77 contributors meeting all of the above criteria. In total, 1432 contributors from 149 projects were selected as our sample to distribute our test automation maturity survey.

Survey design. We reused our previous test automation maturity survey developed in our 2020 study (Wang et al., 2020a). That survey was developed using a knowledge base established upon 18 test maturity models (Wang et al., 2020a, 2019). It has been reviewed by test automation experts and executed with 151 practitioners (coming from above 101 organizations in 25 countries) in the industry. We tailored that survey to fit the needs of our study in this paper. Our survey in this paper contains three parts.

Part 1 contains a consent form, which introduces the survey content, shows the information of principal researchers and organizations who designed the survey, and declaims the research purpose. Only respondents, who consent that they are the right audience of the survey and are willing to participate in the research, are directed to Part 2, otherwise, the survey is closed.

Part 2 presents 16 test automation maturity questions (Table 6). These maturity questions state standard best practices for different key areas of test automation. We designed that respondents answer each maturity question using an ordinal scale: 1- strongly disagree, 2 - disagree, 3 - slightly disagree, 4 - slightly agree, 5 - agree, 6 - strongly agree. The higher agreement selected in the scale reveals that test automation practices carried out in respondents' organizations are more close to the standard best practice stated in a maturity question. Besides, 'no answer' options were provided. Respondents were allowed to leave comments in a free-text field situated at the end.

Part 3 presents several background questions to the project and respondent profile. The purpose is to check the background of respondents and understand in which situations test automation practices are carried out in the CI context of selected projects.

Survey distribution. Our survey was hosted by an online survey tool called LimeSurvey[3] and it was available to invitees (1432 contributors from 149 projects) from 18th October 2020 to 18th January 2021. We sent out personalized email invitations to invitees on 18th October 2020. Later, two rounds of reminder emails were sent out respectably in November and December 2020. Invitees were asked to complete our survey online. The participation was voluntary and anonymous. The survey and its host tool LimeSurvey adhere to GDPR. Invitees were allowed to withdraw their responses at any time. To attract invitees, we established reward mechanisms. When our survey finished, we have sent an individual report that summarizes a snapshot of all survey responses to each respondent. We selected five lucky respondents and gave each a 50-euro Amazon gift card.

[3] https://www.limesurvey.org/.

Table 6
Test automation maturity questions (Wang et al., 2020a).

SQ1-Strategy.	We have a test automation strategy that defines 'what test scope will be automated to what degree, when, by whom, by which methods, by what test tools, in what kind of environment'.
SQ2-Resources.	We allocate enough resources for test automation, e.g., skilled people, the funding, the time & effort, test environment with the required software, hardware, or test data for test automation.
SQ3-Roles.	We clearly define roles and responsibilities of stakeholders in test automation.
SQ4-Knowledge.	We are systematically learning from prior projects. We collect and share expertise, good test automation practices, and good test tools for future projects.
SQ5-Competence.	Our test team has enough expertise and technical skills to build test automation based on our requirements.
SQ6-Tools.	We currently have the right test tools that best suit our needs.
SQ7-Test environment.	We have control over the configuration of our test environment.
SQ8-Guidelines.	We have guidelines on designing and executing automated tests. Those guidelines include, e.g., coding standards, test-data handling methods, specific test design techniques to create test cases, processes for reporting and storing test results, the general rules for test tool usage, or information on how to access external resources.
SQ9-Prioritization.	We effectively prioritize and schedule automated tests for the execution.
SQ10-Results.	We are capable to manage and integrate test results collected from different sources (e.g., different test tools, test levels, test phases) into a big picture, and then report useful information to the relevant stakeholders.
SQ11-Process.	We organize our test automation activities in the stable and controllable test process.
SQ12-SUT.	Our Software Under Test enables us to conduct our test automation, e.g., maturity, running speed, or testability of our Software Under Test is not a problem for our test automation.
SQ13-Measurement.	We have the right metrics to measure and improve our test automation process.
SQ14-Testware.	Our testware (e.g., test cases, test data, test results, test reports, expected outcomes, and other artifacts generated for automated tests) is well organized in a good architecture and it is easy to be maintained.
SQ15-Efficient&Effective.	We create automated tests that are able to produce accurate and reliable results in timely fashion.
SQ16-Satisfaction.	We create automated tests can meet the given test purposes and consequently bring substantial benefits for us, e.g., better detection of defects, increase test coverage, reduce test cycles, good Return on Investment, better guarantee product quality.

In total, we received 43 responses from 41 projects to our survey. The response rate counted for project level is 27.5% (41 out of 149 projects). Since our study in this paper aimed to make observations on the project level, the response rate of our survey also is counted at the project level.

Response quality control. To control the quality of our survey responses, we consulted the online survey standard from Ganassali (2008). To improve the overall quality of our survey, we removed six survey responses from our pool. These six responses either had the same answer on over half of all maturity questions, or had selected 'no answers' on over five maturity questions. As such, a final pool of 37 responses from 37 projects was finally built for our study in this paper.

Project and respondent profile. Table 7 depicts the project and respondent profile of 37 projects in our final pool. We see that test automation coverage is rather high in these projects. Approximately 75% of the projects automated over 50% of test cases. 37.8% of the projects automated at least 90% of test cases. All projects automated unit tests and above half of the projects automated integration tests, while automated tests in other levels take a relatively small proportion in these projects. All respondents are internal contributors and most of them are developers (78.4%) in the projects. Many respondents (73%) have been working on test automation in CI context of these projects for 1–5 years.

Table 7
Project and respondent profile.

	Response
% of automated test cases:	
<10%	2 (5.4%)
11%–50%	4 (10.8%)
51%–90%	14 (37.8%)
>90%	14 (37.8%)
We do not measure	3 (8.1%)
Test-level automation[a]:	
Unit	37 (100%)
Integration	19 (52.8%)
System	6 (16.7%)
Acceptance	2 (5.4%)
Performance	3 (8.1%)
Regression	13 (36.1%)
GUI	7 (19.4%)
Stress	1 (2.8%)
Current role in projects:	
Test Lead/Manager/Director	5 (13.5%)
Developers	29 (78.4%)
Testers	3 (8.1%)
Years of working test automation in CI context of the projects:	
6–10 years	7 (18.9%)
1–5 years	27 (73.0%)
Less than 1 year	3 (8.1%)

[a]This is a multiple choice question.

Response overview. Fig. 2 shows the overview of responses to maturity questions. Agreed responses (slightly agree–strongly agree) are stacked to the right of a vertical baseline on '0%' on the *x*-axis. Disagreed responses (slightly disagree–strongly disagree) are stacked to the left of the same baseline. Note that, since 'no answer' exists for certain questions, the total percentage of each question may be not equal to 100%. "SQ6-Tools" and "SQ5-Competence" got the highest percentage of agreed responses (88% and 86%), suggesting that, most projects have the right test tools and competent test professionals to conduct test automation in their current CI context. "SQ9-Prioritization" and "SQ10-Results" have the highest percentage of disagreed responses (51%), suggesting that, many projects still are not capable to effectively prioritize automated tests and handle test automation results in their current CI context. In our previous study (Wang et al., 2020b) that surveyed 151 practitioners using the same survey, "SQ5-Competence" also got the highest percentage of agreed responses (85%), while "SQ8-Guidelines" (47%) got the highest percentage of disagreed responses indicating there is a lack of guidelines on designing and executing automated tests in general.

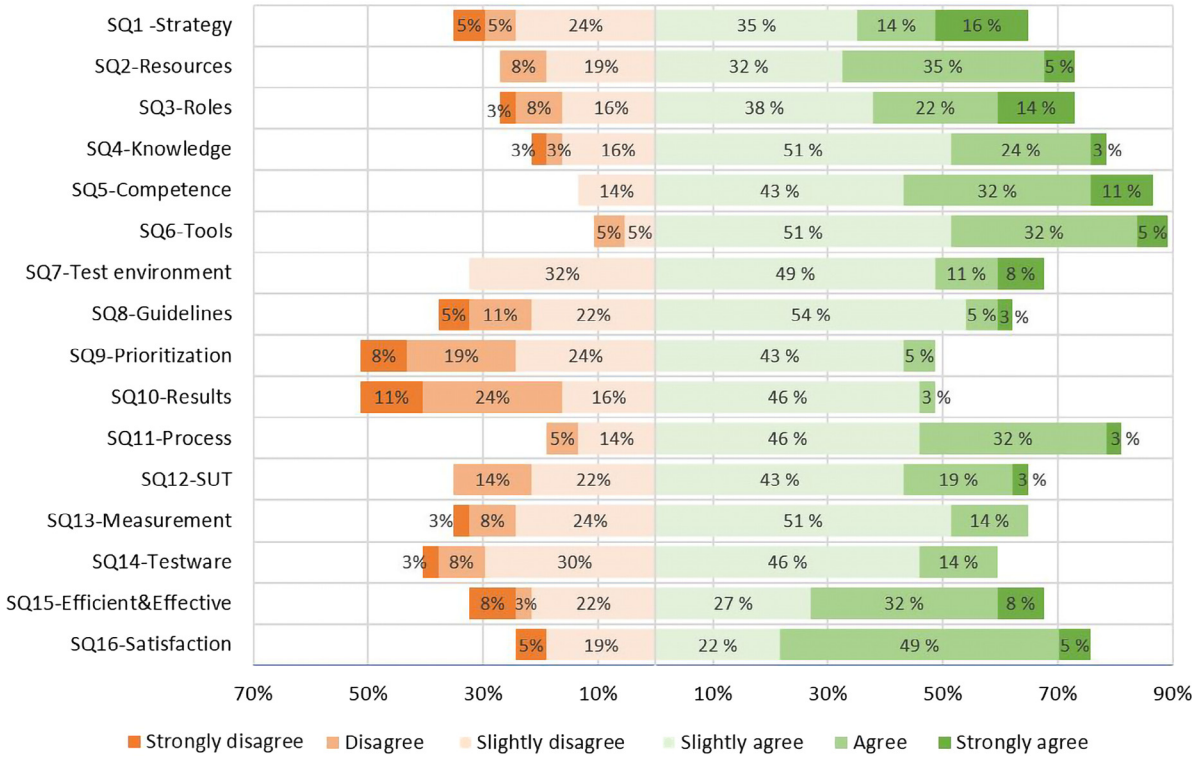


Fig. 2. Survey response.

As described in the above survey design related paragraphs, accompanying with maturity questions, there is a free-text field situated to allow respondents to leave free comments. However, there was no respondent leave free comments in that field.

3.4. Project data collection

We collected data from 37 projects in our final pool. Two snapshots were set on each project. Snapshot 2 represents the state of the project on 18th January 2021 (the date when our test automation survey closed). Snapshot 1 represents the state of the project on 18th January 2020 (one year before Snapshot 2). We studied each project with its development practices between these two snapshots. Two snapshots had a one-year interval. The reason to set one-year interval is: short interval based data (e.g., three-month-interval) may be insufficient to provide a stable overview of the project's release cycles, test automation effort, and product quality; long interval based data (e.g., two-year-interval) may not correspond to the project's current state of test automation practices (that we surveyed in our test automation maturity survey). Table 8 shows the metrics we used to measure variables in our conceptual model (Fig. 1). To collect metric data, we collect survey responses, and mined each project's GitHub repository, CI repository, and issue track system. Table 9 presents summary statistics for collected metric data.

"Test automation maturity" was measured by the total score of all maturity questions in our survey. For each maturity question, the answer on an ordinary scale was recorded in a score of 1–6 (from strongly disagree to strongly agree); 'no answer' converted to '0'. The total score sums the score of all maturity questions. The higher total score represents that, test automation practices within a respondent's organization are more close to standard best practices in the literature suggesting a higher level of test automation maturity.

The metric for "Product quality" was "Defect density" that is widely used by prior scholars (Agrawal and Chari, 2007; Harter et al., 2000). Defect density measures the defects related to the product size — the larger value of defect density imply a higher frequency of defects per unit of a product, i.e., lower product quality. In our study, defect density was computed as the cumulative number of post-release defects (reported by contributors and users) divided by product size at Snapshot 2. Post-release defects were used as they are the ones neglected by CI practices, and thus, can be used to observe how CI practices (with test automation maturity) is effective to improve product quality (Rwemalika et al., 2019). In step 2 of our project selection process, we have selected projects that "use naming convention (e.g., bug, defect, fault, error, flaw, and other synonyms) to label defects" in their issue tracker system, see Table 5 in Section 3.2. Thus, to compute the cumulative number of post-release defects reported between two snapshots for each project, we count the number of issues assigned with defects-related labels after releases between two snapshots in each project's issue tracker system.

In our conceptual model, 'Test automation effort" is the combination of test automation development effort and execution effort. Separate metrics were used for measuring test automation development effort and execution effort. Each project's test automation development effort was measured by increased LOC (without comments and blanks) of test automation between two snapshots. To measure software development effort, software engineering researchers have discovered different metrics: LOC, Person-days (the time in days required for one person to complete a task), active-days (the number of days contributors make a commit) (Jorgensen, 1995; Shihab et al., 2013; Boehm et al., 2000; Bergeron and St-Arnaud, 1992). In this paper, LOC was used as it is more widely used than other metrics for measuring the development effort of open source projects (Jorgensen, 1995; Shihab et al., 2013; Boehm et al., 2000; Bergeron and St-Arnaud, 1992). Besides, to measure test automation execution effort for each project, the total time to execute all automated tests in CI builds between two snapshots was computed. For each project, we mined test log files to collect the total time

Table 8
Variable metrics.

Observed variable	Metric
Test automation maturity	Total score of all maturity questions in part 2 of our test automation maturity survey
Product quality	Defect density = the cumulative number of post-release bugs between two snapshots/KLOC[a] of production at Snapshot 2
Test automation effort	Development effort (in seconds) = Increased LOC (without comments and blanks) of test automation between two snapshots Execution effort (in seconds) = Total time to run all automated tests in CI builds between two snapshots
Release cycle	The number of releases between two snapshots
Control variable	Metric
Product size	LOC[b] (without comments and blanks) of the current product at Snapshot 2
Product complexity	Average Cyclomatic complexity number on all coding files of the current product at Snapshot 2
Product age	Product age in days since the date of the first commit to the date of Snapshot 2
Team size	The average number of contributors per month between two snapshots
Product popularity	The number of forks at Snapshot 2
Integration frequency	The number of CI builds between two snapshots

[a]KLOC:thousands of lines of code.
[b]LOC:lines of code.

Table 9
Summary statistics on variable related metric data.

Variable	Mean	Median	St. Dev.	Min	Max
Test automation maturity	61.865	64	12.383	30	84
Defect density (meas.Product quality)	0.566	0.399	0.699	0.037	3.737
Test automation effort:					
Development effort	6,852.595	1614	21,120.510	17	127,113
Execution effort	2,517,660	299341	8,079,574	3,078	48,158,813
Number of releases (meas.Release time)	8.757	6	7.429	1	29
Product size	229,234.200	77,166	398,647.300	3,479	1,868,078
Product complexity	1.997	1.8	0.639	1.200	4.600
Product age	3,765.378	3762	1,051.079	578	6,782
Team size	5.135	3	5.213	1	27
Product popularity	5,212.838	398	15,225.550	10	86,000
Integration frequency	739.9	506	777.165	28	2620

```
2515 [INFO] Running org.fao.geonet.repository.specification.SettingSpecTest
2516 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.095 s - in
     org.fao.geonet.repository.specification.SettingSpecTest
2517 [INFO] Running org.fao.geonet.repository.specification.UserGroupSpecsTest
2518 [INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.182 s - in
     org.fao.geonet.repository.specification.UserGroupSpecsTest
2519 [INFO] Running org.fao.geonet.repository.specification.UserSpecsTest
2520 [INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.118 s - in
     org.fao.geonet.repository.specification.UserSpecsTest
```

Fig. 3. The standard output of Maven tests from Core-geonetwork.

of all CI builds (occurred between two snapshots) for executing automated tests. Maven standard output provided a convenient way to do that. Fig. 3 shows an example from the project Core-geonetwork,[4] with Maven standard output, the execution time is printed out for units of automated tests. In the end, to combine test automation development effort and execution effort into test automation effort, we did the data normalization, as the scale of execution effort is different from the scale of development effort, see Tables 8 and 9. Test automation development effort and execution effort were normalized on a scale of 1–100 separately for each project. The sum of the normalized development effort and execution effort was computed as "Test automation effort" for each project.

'Release cycle' was measured by the number of releases between two snapshots — the more number of releases indicates shorter release cycles for a project.

All control variables were measured at Snapshot 2 that represents the current state of a project. Product size was measured by LOC developed in the current product excluding comments and blanks. The use of LOC is in line with prior scholars (Agrawal and Chari, 2007; Harter et al., 2000). Average Cyclomatic complexity number (McCabe, 1976) was computed (with a tool Lizard[5]) to determine product complexity. It measures the number of linearly independent paths in source code (McCabe, 1976). A lower average Cyclomatic complexity number indicates the lower product complexity. As contributors may join or leave at any phases of

4 https://travis-ci.org/github/geonetwork.

5 https://github.com/terryyin/lizard.

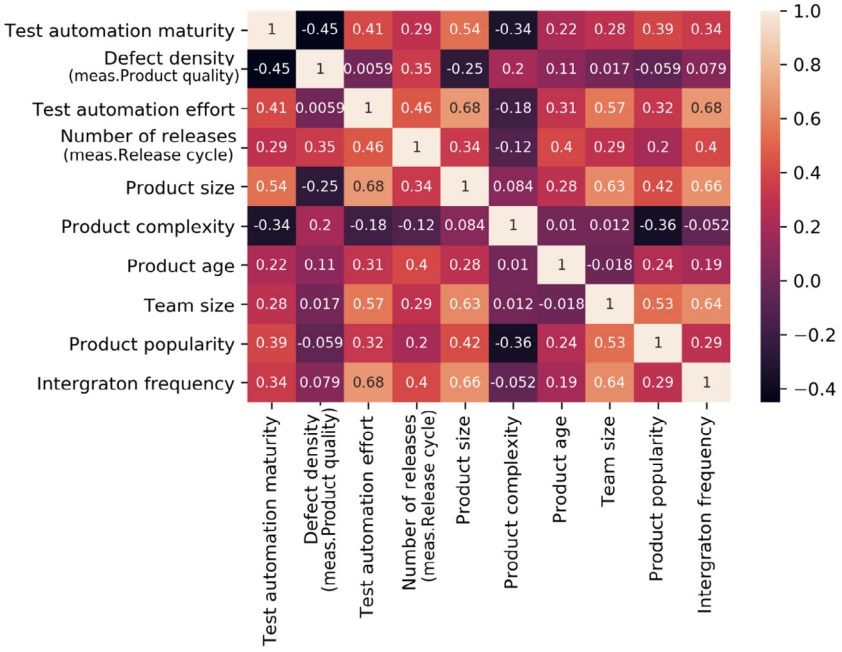


Fig. 4. Correlation Matrix.

open source projects (Vasilescu et al., 2015), a project's team size was measured as the average number of contributors per month between two snapshots. Product popularity was measured by the number of forks of a project at Snapshot 2 - a large number of forks indicates higher product popularity. Integration frequency was measured by the number of CI builds between two snapshots — more number of CI builds indicates higher integration frequency.

3.5. Data analysis

We first observed if there is a correlation between variables in our conceptual model. Fig. 4 presents the correlation matrix of metric data on our variables. Spearman method was used to compute the correlation matrix, as it can measure the strength of both linear and non-linear association between two variables (Croux and Dehon, 2010). As uni-variate correlations are not solid statistical evidence, it is imperative to run regression analysis to further study our conceptual model with metric data.

To validate our conceptual model (Fig. 1) that hypothesized the relationships among our observed variables, we developed several empirical models:

1. EM1. Product quality = f(Test automation maturity, product size, Product complexity, Product popularity, Product age, Team size, Integration frequency)
2. EM2. Test automation effort = f(Test automation maturity, Product quality, Product size, Product complexity, Product age, Team size, Integration frequency)
3. EM3. Release cycle = f(Test automation maturity, Product quality, Test automation effort, Product size, Product complexity, Team size, Integration frequency)

Each empirical model addresses at least one hypothesis in our conceptual model. Table 10 shows which hypothesis(es) each empirical model addresses. We applied multiple regression on our metric data of our variables to study our empirical models. However, before starting regression analysis, we did further observations on metric data of our variables.

We used residual plots to investigate if our metric data is suitable for linear regression analysis on EM1–EM3, see Fig. 5.

Table 10
Empirical model v.s. hypothesis.

Empirical model	Hypothesis
EM1	H1. Higher levels of test automation maturity are associated with higher product quality.
EM2	H2. Higher levels of test automation maturity are associated with increased test automation effort; H3. Higher product quality is associated with increased test automation effort.
EM3	H4. Higher levels of test automation maturity are associated with longer release cycles; H5. Higher product quality is associated with longer release cycles; H6. Increased test automation effort is associated with longer release cycles.

We observed typical problems including non-linearity, outliers, and heteroskedasticity (Astivia and Zumbo, 2019). One can see, all residual plots are in a non-random pattern, suggesting the presence of non-linearity. Shapiro–Wilks Normality Test (Shapiro and Wilk, 1965) also rejected the presence of linearity in our empirical models. In all residual plots, we can see some outliers (numbered) that are far away from the red line. The heteroskedasticity is presented, as the red line is not straight, the distribution of predicted values throughout the red line is not equal, and the residuals seem to increase with the fitted values increase in each residual plot.

In light of the above observations, Box–Cox Transformation was used to transform our data into normality and mitigate the effect of heteroskedasticity. Box–Cox Transformation can reduce anomalies in the data to ensure the usual assumption for a linear model hold (Box and Cox, 1964). For a given set of dependent and independent variables, it identifies the right specification by transforming the dependent variable (Vélez et al., 2015; Box and Cox, 1964). Let $y = (y_1, y_2, \ldots, y_n)'$ be the given data on which the Box–Cox Transformation is to be applied, the Box–Cox transformation on y is:

$$y \Rightarrow (y^{\lambda} - 1)/\lambda$$

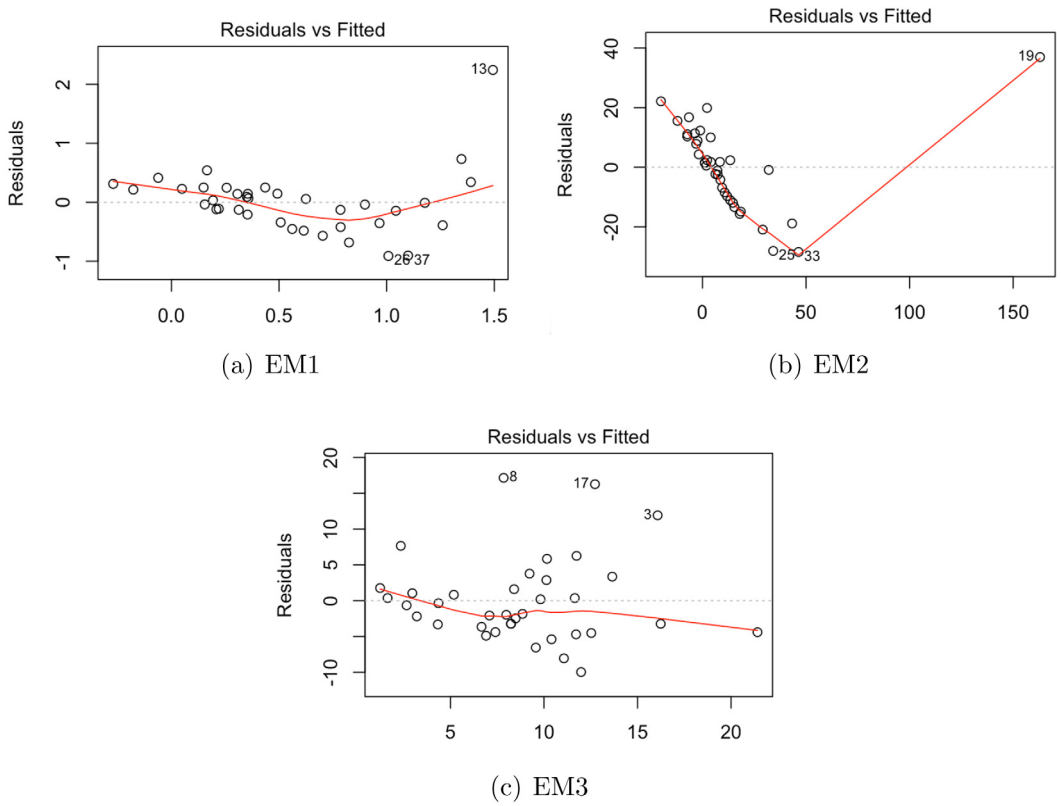


Fig. 5. Residuals & Fitted plots for empirical model EM1, EM2, EM3.

y^{λ} is the λ-transformed data (Box and Cox, 1964). The maximum likelihood estimate of λ is computed to identify the transformation for the right specification (Box and Cox, 1964). All values of λ are considered to select the optimal value for the given data y (Vélez et al., 2015; Box and Cox, 1964). The optimal value is the one that can result in the best approximation for the normal distribution (Vélez et al., 2015; Box and Cox, 1964).

In our paper, we used R function boxcox()[6] to estimate λ for our empirical model EM1–EM3. Fig. 6 plots the log-Likelihood as a function of λ values for each model. We computed that 0.083, −0.25, 0.073 are optimal λ values that maximize the log-likelihood for EM1, EM2, and EM3. Hence, with Box–Cox Transformation, our empirical models for multiple regression analysis were defined as follows:

- EM1. $((\text{Product quality})^{0.083} - 1)/0.083 = \beta_{Q0} + \beta_{Q1}$(Test automation maturity) + β_{Q2}(Product size) + β_{Q3}(Product complexity) + β_{Q4}(Product popularity) + β_{Q5}(Product age) + β_{Q6}(Team size) + β_{Q7}(Integration frequency)
- EM2. $((\text{Test automation effort})^{-0.25} - 1)/ - 0.25 = \beta_{C0} + \beta_{C1}$(Test automation maturity) + β_{C2}(Product quality) + β_{C3}(Product size) + β_{C4}(Product complexity) + β_{C5}(Product age) + β_{C6} (Team size) + β_{C7}(Integration frequency)
- EM3. $((\text{Release cycle})^{0.073} - 1)/0.073 = \beta_{R0} + \beta_{R1}$(Test automation maturity) + β_{R2}(Product quality) + β_{R3}(Test automation effort) + β_{R4}(Product size) + β_{R5}(Product complexity) + β_{R6}(Team size) + β_{R7}(Integration frequency)

4. Results

We present our findings in three aspects (as defined in our conceptual model): Effect of test automation maturity on product quality, Effect of test automation maturity and product quality on test automation effort, and Effect of test automation maturity and product quality on release cycle. Each aspect's findings are described below.

4.1. Effect of test automation maturity on product quality

Table 11 presents the regression result of EM1. **Our hypothesis "H1. Higher levels of test automation maturity are associated with higher product quality" was supported.** As shown in Table 11, even though under the control of many other variables, "Test automation maturity" (coefficient = −5.070e−02, p-value = 0.000624) is highly significantly associated with "Defect density" (that measures "Product quality"). The negative association here indicates that, higher levels of test automation maturity are highly significantly associated with lower defect density–higher product quality.

Moreover, we found that, the effect of test automation maturity is more significant than product size, product complexity, product popularity, product age, team size, and integration frequency on product quality. This can be seen from Table 11, among all variables, "Test automation maturity" is most significantly associated with "Defect density". Besides, we found that, "Project age" (coefficient = 2.790e−04, p-value = 0.056590) has a positive association with "Defect density" suggesting that old projects exhibit high defect density–lower product quality.

4.2. Effect of test automation maturity and product quality on test automation effort

Table 12 presents the regression result for EM2. **Our hypothesis "H2. Higher levels of test automation maturity are associated with increased test automation effort" was rejected**. As shown in Table 12, even though "Test automation maturity" has a positive coefficient (1.503e−03), it has no significant

[6] https://www.rdocumentation.org/packages/EnvStats/versions/2.4.0/topics/boxcox.

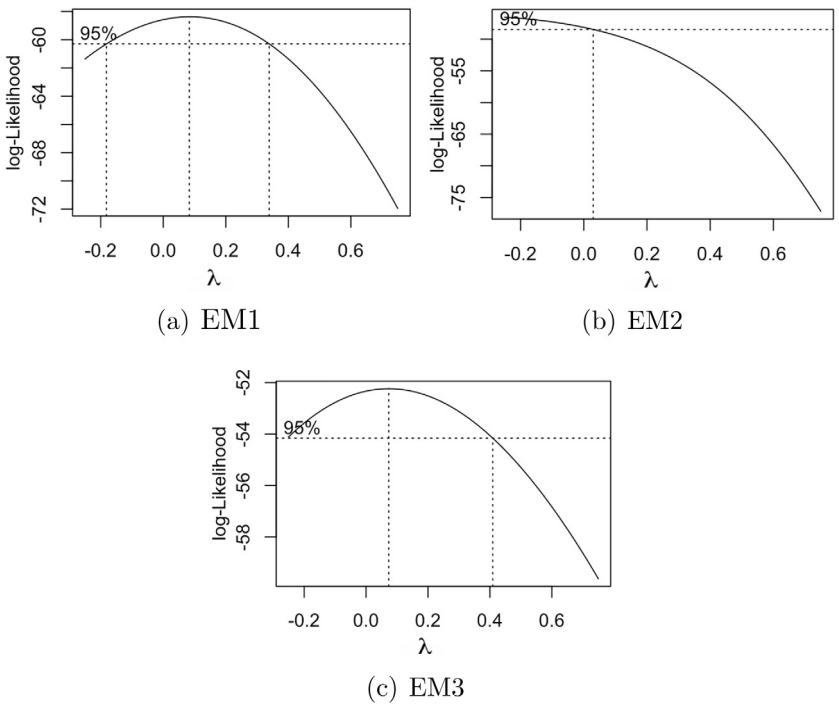


Fig. 6. Box–Cox Transformation: log-likelihood as a function of λ values.

Table 11
Regression results of EM1: Defect density (meas.Product quality) as dependent variable.

	Estimate	Std. Error	statistic	*p*-value
(Intercept)	7.000e−01	1.069e+00	0.655	0.517679
Test automation maturity	−5.070e−02	1.322e−02	−3.835	0.000624***
Product size	−3.070e−07	6.437e−07	−0.477	0.637045
Product complexity	3.636e−02	2.708e−01	0.134	0.894115
Product popularity	1.094e−05	1.729e−05	0.633	0.531921
Project age	2.790e−04	1.405e−04	1.986	0.056590 .
Team size	3.435e−02	6.617e−02	0.519	0.607657
Integration frequency	1.788e−04	3.030e−04	0.590	0.559683

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Table 12
Regression results of EM2: Test automation effort as dependent variable.

	Estimate	Std. Error	statistic	*p*-value
(Intercept)	1.222e+00	6.188e−01	1.974	0.0579 .
Test automation maturity	1.503e−03	8.518e−03	0.176	0.8611
Defect density (meas.Product quality)	−2.867e−02	1.320e−01	−0.217	0.8296
Product size	1.810e−07	3.201e−07	0.565	0.5761
Product complexity	−2.485e−01	1.219e−01	−2.038	0.0508 .
Product age	4.323e−05	7.761e−05	0.557	0.5818
Team size	5.299e−02	2.529e−02	2.096	0.0449*
Integration frequency	8.549e−05	1.233e−04	0.693	0.4936

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

association (*p*-value =0.8611) with dependent variable "Test automation effort". **The regression results of EM2 also rejected our hypothesis "H3. Higher product quality is associated with increased test automation effort".** As shown in Table 12, there was no significant association between "Defect density" and "Test automation effort", as the *p*-value of "Defect density" (0.8296) is close to 1. The above results indicate that, increased test automation effort caused by improving the level of test automation maturity and product quality is not statistically evidenced.

Moreover, we found that, the effect of product complexity and team size is more significant than the effect of test automation maturity and product quality on test automation effort. This can be seen from Table 12, "Product complexity" (*p*-value = 0.0508) and "Team size" (*p*-value = 0.0449) have a significant association with dependent variable "Test automation effort", while "Test automation maturity" (*p*-value = 0.8611) and "Defect density"(*p*-value = 0.8296) do not have. "Product complexity" (coefficient = −2.485e−01) has a significant negative association with "Test automation effort", suggesting that, more complex products cost less test automation effort. "Team size" (coefficient = 5.299e−02) has a significant positive association with "Test automation effort", suggesting that, larger teams spend more test automation effort.

Table 13
Regression results of EM3: Number of releases (meas.Release cycle) as dependent variable.

	Estimate	Std. Error	statistic	*p*-value
(Intercept)	−1.194e+00	1.340e+00	−0.891	0.38024
Test automation maturity	4.135e−02	1.663e−02	2.486	0.01891*
Defect density (meas.Product quality)	8.913e−01	2.531e−01	3.521	0.00144**
Test automation effort	7.101e−03	1.031e−02	0.689	0.49643
Product size	5.973e−08	7.804e−07	0.077	0.93951
Product complexity	9.987e−03	2.805e−01	0.036	0.97184
Team size	−5.701e−02	6.895e−02	−0.827	0.41515
Integration frequency	3.453e−04	3.271e−04	1.056	0.29989

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

4.3. Effect of test automation maturity and product quality on release cycle

Table 13 presents the regression result of EM3. **Our hypothesis "H4. Higher levels of test automation maturity are associated with longer release cycles" was rejected.** One can see that, "Test automation maturity" (coefficient =4.135e−02, *p*-value = 0.01891) presents a significant positive association with dependent variable "Number of releases". This suggests that, achieving higher levels of test automation maturity have a positive effect on release cycle acceleration.

Our hypothesis "H5. Higher product quality is associated with longer release cycles" was supported. As shown in Table 13, "Defect density" (coefficient= 8.913e−01, p-value= 0.00144) has a significant positive association with dependent variable "Number of releases". This suggests that the direct effect of product quality on release time is negative — better product quality exists in longer release cycles.

The regression result of EM3 rejected our hypothesis "H6. Increased test automation effort is associated with longer release cycles". We did not identify a significant association between "Test automation effort"(*p*-value = 0.49643) and "Number of releases', see Table 13. The result reveals that test automation effort does not seem to impact release time.

5. Discussion

We present summary and discussion of study findings in Section 5.1, explore the implications for researchers in Section 5.2 and implications for practitioners in Section 5.3, and examines limitations in Section 5.4.

5.1. Summary and discussion of study findings

This paper intends to answer a research question "Do higher levels of test automation maturity lead to better product quality without increased test automation effort and release time in the CI context of open source projects?". To answer our research question, we built a conceptual model, using literature, to hypothesize the relationships between Test automation maturity, Product quality, Test automation effort, and Release cycle. We studied 37 open source java projects to test our conceptual model. Our study findings are formulated in three aspects (as defined in our conceptual model): Effect of test automation maturity on product quality, Effect of test automation maturity and product quality on test automation effort, and Effect of test automation maturity and product quality on release cycle. We first discuss our findings in each aspect and next answer our research question.

Effect of test automation maturity on product quality. We found that, higher levels of test automation maturity (assessed by standard best practices in the literature) are highly significantly associated with higher product quality in the CI context of observed open source projects. This finding suggests that standard best practices in the literature can ensure the effectiveness of test automation in detecting defects and thus assure product quality in the CI context of open source projects. Our finding is consistent with findings made by prior scholars (Wang et al., 2020b; Puri-Jobi, 2015; Hilton et al., 2017), who reported that better product quality can be achieved after adopting certain standard best practices for test automation maturity improvement in the CI context. However, our study examined a boarder range of standard best practices, as it was based on extensive literature reviews for standard best practices of test automation in our early studies (Wang et al., 2019, 2020a).

To extend prior research on this topic, when analyzing the relationship between test automation maturity and product quality, we involved several control variables: product size, product complexity, product popularity, product age, team size, and integration frequency. Our study is the first attempt to evidence that, the effect of test automation maturity is more significant than product size, product complexity, product popularity, product age, team size, and integration frequency on product quality in the CI context of open source projects. It again confirms the importance of improving the level of test automation maturity in the CI context of open source projects and also the positive impact of standard best practices on that. Furthermore, we found that, old projects exhibit lower product quality. This finding conflicts with the observation of prior scholars (Harter et al., 2000), who examined that, older products have a lower chance of exposure to defects. The difference might come from the approach to compute defect density (we used post-release defects while they used pre-release defects), types of projects (we observed open sources projects while they observed closed source projects), or varied project technologies (we selected projects that are written in Java and use CI tools while their sampled projects were different). Besides, many studies (Cotroneo et al., 2016; Ibrahim et al., 2012; Au et al., 2009) have observed that, larger teams may insert more defects to a software product because more people means a higher occurrence chance of defects. Yet, our study did not find a significant association between team size and product quality. The possible explanation is that, compared to prior studies, our study included more variables — as such, the effect of team size on product quality is not that significant under the control of other variables.

Effect of test automation maturity and product quality on test automation effort. Our study did not find a significant association between test automation maturity/product quality and test automation effort in the CI context of observed open source projects, when controlling for product size, product complexity, product age, team size, and integration frequency. This reveals that, increased test automation effort caused by improving the level of test automation maturity and product quality is not evidenced. Our finding conflicts with prior studies' viewpoint (Lin et al., 2020; Hilton et al., 2017; Lee and Hwang, 2012; Williams et al., 2009; Kasurinen et al., 2010), which states that, at higher levels of test automation maturity and product quality, there is increased effort to develop and execute rigorous automated tests.

Our finding confirms the other viewpoint, which states the effort savings may offset the invested effort when using standard best practices to perform rigorous test automation (Berner et al., 2005; Kumar and Mishra, 2016; Puri-Jobi, 2015; Tosun et al., 2009). Based on many sources (Graham and Fewster, 2012; Garousi and Varma, 2010; Garousi and Elberzhager, 2017), test automation maturity improvement is an investment that needs efforts to implement, while the return is not linear — cost savings would be achieved after a certain period of time and then increase with the time grows. That could be the explanation for the existence of prior two different viewpoints. Studies (Puri-Jobi, 2015; Tosun et al., 2009; Berner et al., 2005) observed that, effort savings (from the reuse of quality test artifacts, effective strategic planning and test prioritization, structured test automation process, the usage of development guidelines, improved SUT testability, and increased competency of test professionals) exceed the invested effort when test automation maturity improvement programs were fully implemented. On the contrary, studies (Lee and Hwang, 2012; Williams et al., 2009) observed increased test automation effort when a standard test process was initialized or not fully implemented, while studies (Lin et al., 2020; Hilton et al., 2017) did not consult cost-savings with respect to the progress of test automation maturity improvement.

Concerning control variables, we found that the effect of product complexity and team size is more significant than test automation maturity and product quality on test automation effort. This finding is not presented by prior scholars. Many prior studies (Ståhl et al., 2017; Banker, 1992; Zhao et al., 2017; Shahin et al., 2017) observed that larger teams would spend more test automation effort. Our study evidenced that their observation is valid in the CI context of open source projects. Besides, our study identified that more complex products spend less test automation effort. This finding conflicts with the observation of prior scholars (Harter et al., 2000; Agrawal and Chari, 2007), who have empirically proofed that it would take more effort to develop and execute automated tests on more complex software products. One possible explanation is that for more complex products developers automate fewer tests due to complexity (Garousi and Mäntylä, 2016).

Effect of test automation maturity and product quality on release cycle. Our study empirically evidenced that, higher levels of test automation maturity (assessed by standard best practices in literature) are significantly associated with shorter release cycles, controlling for product size, product complexity, team size, and integration frequency. Our finding is consistent with prior studies' observation, which views short release cycles can be enabled by improving the level of test automation maturity (Wang et al., 2020b; Collins and de Lucena, 2012; Berner et al., 2005; Kumar and Mishra, 2016; Puri-Jobi, 2015). On the contrary, our finding conflicts with the other viewpoint, which states, release time must increase against improvements of the level of test automation maturity (Kasurinen et al., 2010; Williams et al., 2009). Such a prior viewpoint explained that, developing and executing rigorous automated tests (using standard best practices) to ensure product quality requires considerable effort, so that the elapsed time is added for product development and thus product releases are delayed (Kasurinen et al., 2010; Williams et al., 2009). Yet, in our study, the association between test automation effort and release cycle was not statistically evidenced. This suggests test automation effort does not seem to affect release time in the CI context of open source projects.

Prior studies have empirically observed that, with short release cycles, even though the testing period is short, defects were fixed early so that product quality is improved in the CI or CI like context (Wang et al., 2020b; Berner et al., 2005; Puri-Jobi, 2015; Collins and de Lucena, 2012). Our finding conflicts with the observation of these prior studies. We observed that the direct effect of product quality on release time is negative – better product quality exists in longer release cycles – in the CI context of observed open source projects. One possible explanation is, the number of releases depends on the evolving set of features that may insert defects. Scholars (Greer and Ruhe, 2004; Ruhe and Saliu, 2005) observed that, as a software release is a collection of new and/or changed features, more new and/or changed features would enable more frequent releases; However, since features are developed in codes and "defects are coding caused", more new and/or changed features would insert more defects. Another potential reason could be there is a threshold for the length of release time. That is, there needs sufficient time to make releases in the CI context. Based on a case study (Hamdan and Alramouni, 2015) that measured product quality before and after applying CI in the development of a software system, it usually took some time to "plan a release" (define release goal, preview the product backlog, and testing a release) before publishing it to users, and hence, too short release cycles in which releases are not planned adequately may make products to be easily exposed to post-release defects (that was used to measure product quality in our research). Also, if the release time is too short, the testing period is too short so that test automation cannot reach the proper deep and cover the right scope, and thus, product quality suffers (Shahin et al., 2017; Hamdan and Alramouni, 2015).

Answers on our research question. Our observations suggest that, in the CI context of open source projects, a potential benefit of improving the level of test automation maturity against standard best practices in the literature is product quality improvement and release cycle acceleration, while increased test automation effort caused by improving the level of test automation maturity and product quality is not evidenced. Thus, we draw a conclusive answer for our research question: higher levels of test automation practices can lead to better product quality without increased release time and test automation effort in the CI context of open source projects.

5.2. Implications for researchers

From a theoretical standpoint, our results suggest several directions for theory development in this research domain. First, our study found that using standard best practices in the literature to conduct test automation can enable CI success in open source projects. Researchers in this domain could explore how to adopt each standard best practice (mentioned in our survey) more deeply. Future research can build on these results to develop test automation maturity models for the CI context. Second, we noted some different observations between prior studies and our study around some control variables. In particular, many studies observed that larger teams tend to insert more defects to a software product, while our study did not find a significant association between team size and product quality; Prior studies observed that more complex products need more effort to develop and execute automated tests, whereas our observation is different; Prior studies observed that older products expose fewer defects due to the increased experience of the team, while we observed that old projects exhibit lower product quality. Due to the scope of our study, we only noted different observations on those control variables and explored possible reasons leading to that. Future efforts to provide theoretical explanations for such different observations would be meaningful. Third, our study demonstrates a novel investigation of CI success from a test automation maturity based view. Prior studies have actively researched on CI process as the determinant of CI success, for example, the integration flow, building frequency, integration serialization and batching, building status communication (Ståhl

and Bosch, 2014; Ghaleb et al., 2019; Ståhl et al., 2017). Future research could also include the view of test automation maturity as the determinant in CI success literature.

Our study demonstrates a triangulated method, that surveyed practitioners and mined project data from multiple repositories (GitHub repository, Travis CI repository, issue track system), to study test automation practices in the CI context of open source projects. Other researchers can consult this triangulated method to study other test automation topics in the CI context, e.g., the effectiveness of test automation, execution efficiency of test automation, or productivity outcomes of adopting CI with test automation.

In our study, we studied many variables and collected metric data on these variables: test automation maturity, product quality, test automation effort (test automation development effort and execution effort), release cycle, product size, product complexity, product age, product popularity, team size, and integration frequency. For researchers who observe similar variables in their research, our study would provide the reference on metric selection and data collection on their observed variables. We clearly documented our data collection process. This would give others a hint about how to mine relevant metric data on these variables from open source project repositories.

Moreover, our dataset contains test automation maturity survey responses, project data, test suite, and test logs of 37 open source java projects. We have made our dataset publicly available. Our dataset could be used by researchers to study other test automation topics. For example, test suite and test logs of these projects can be used for test prioritization or test case selection related studies; Test code quality related studies can make observations on test suites of these projects; Test automation survey responses and test logs can be studied together to explore the state of art and practice of test automaton of open source projects; Quality assurance outcomes related studies can use the count of reported bugs of these projects.

5.3. Implications for practitioners

Our study has implications for practitioners who are working on CI with test automation. Our study evidenced that, a potential benefit of improving the level of test automation maturity (using standard best practices in the literature) is product quality improvement and release cycle acceleration in the CI context of open source projects. Knowing this finding would help practitioners to assess and improve their test automation maturity towards CI success in the similar CI context within their organizations. They could assess and improve test automation practices against standard best practices mentioned in our survey. They also can collect similar metric data (used in our study) to observe how test automation maturity affect product quality, test automation effort, and release cycle in their CI context, or analyze costs and benefits of test automation/CI, i.e., good product quality and short release cycles as benefits and test automation effort as costs.

5.4. Limitations

Like any other studies, this study also has some limitations. First, a limitation is that our dataset might not be representative of all open source projects that are adopting test automation in the CI context. We studied 37 open source java projects using Travis CI and Maven. We must select projects in the same programming language, using the same CI server tool, and using the same CI building framework, because different ones may have different performance and structure (that may bias our research results). Although we aimed to select and study the large sample of open source java projects, given our limited resource constraints, it finally did not realize in the end. In the project selection phase, we began with JTec and 20-MAD datasets that contain over 30k open source java projects in total, but in the end only 149 projects met our needs. A large number of projects were excluded as we did not have the access to those projects' private Travis CI repository and issue tracker system. In the survey distribution phase, we also lost the projects in which selected contributors did not answer our survey. Thus, we encourage future research to extend our findings by adding more datasets. For example, extending datasets with open source projects written in other popular programming languages such as Python, C++ and JavaScript, or use other CI server tools (like Jenkins or CircleCI) and building tools (like Gradle).

Second, since our study only examines open source projects, the limitation lies in whether our study results can be generalized to other types of software development contexts that adopt test automation using similar CI tools, for example, closed source software development contexts and large-scale industrial software development contexts. By considering differences, similar experiment studies can be carried out to validate our conceptual model (Section 3.1) in other types of software development contexts. For example, as full-time developers are hired to work on closed source projects, estimating or calculating test automation effort using the metric Person-days is more accurate than the metric LOC used in our study. Another example is, for complex large-scale industrial projects consisting of millions of LOCs, the control variable "Product complexity" may need to be measured by the integration of multiple metrics such as Cyclomatic complexity number (used in our study), Structural coupling, and Logical coupling (Sarkar et al., 2008).

Third, we used "Increased LOC of test automation (without counting of blanks and comments)" as the metric to measure test automation development effort, because this metric is more widely used than other metrics for open source projects (see Section 3.4) and calculating the actual test automation development effort for each project is not feasible. However, we believe that future research is needed to validate our findings with the actual test automation development effort in open source projects and others (e.g., closed source projects and large-scale industrial projects).

Finally, in our conceptual model, we included control variables that affect the relationships between our observed variables. All control variables were identified by prior scholars. Additional control variables may exist in the real industrial context but have not been identified by prior scholars. When new control variables are identified, the replication of this study is required to evaluate findings by including new control variables.

6. Conclusion

In this paper, we have empirically studied the effect of test automation maturity (assessed by standard best practice in the literature) on product quality, test automation effort, and release cycle in the CI context of 37 open source java projects. Our study showed that, in such the CI context, a potential benefit of improving the level of test automation maturity is product quality improvement and release cycle acceleration, while increased test automation effort caused by improving the level of test automation maturity and product quality is not evidenced. Our results suggest that test automation related standard best practices defined in the literature would fit the CI context of open source projects and enable CI success. Our study evidenced some observation of prior studies, resolved the different viewpoints of prior studies, and identified novel research topics to extend the impact and scope of CI and test automation literature, see details in Section 5.1 and Section 5.2. Our recommendation to

practitioners (in the similar CI context) is to utilize standard test automation best practices to improve test automation maturity towards test automation success as well as CI success, see details in Section 5.3.

CRediT authorship contribution statement

Yuqing Wang: Conceptualization, Methodology, Investigation, Formal analysis, Data curation, Writing – original draft. **Mika V. Mäntylä:** Conceptualization, Methodology, Investigation, Writing – review & editing, Supervision, Funding acquisition. **Zihao Liu:** Methodology, Investigation, Data curation, Formal analysis, Validation. **Jouni Markkula:** Methodology, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by TESTOMAT Project (ITEA3 ID number 16032) funded by Business Finland under Grant Decision ID 3192/31/2017, and the foundation of Tauno Tönning (project ID 20210086). This work receives open access funding provided by the University of Oulu including Oulu University Hospital.

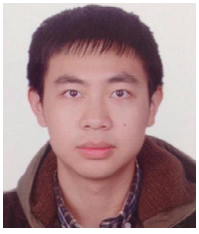
References

Agrawal, M., Chari, K., 2007. Software effort, quality, and cycle time: A study of CMM level 5 projects. IEEE Trans. Softw. Eng. 33 (3), 145–156.
Astivia, O.L.O., Zumbo, B.D., 2019. Heteroskedasticity in multiple regression analysis: What it is, how to detect it and how to solve it with applications in R and SPSS. Pract. Assess. Res. Eval. 24 (1), 1.
Au, Y.A., Carpenter, D., Chen, X., Clark, J.G., 2009. Virtual organizational learning in open source software development projects. Inf. Manage. 46 (1), 9–15.
Banker, R.D., 1992. Software Complexity and Software Maintenance Costs. Center for Information Systems Research, Sloan School of …, Cambridge, Mass..
Bergeron, F., St-Arnaud, J.-Y., 1992. Estimation of information systems development efforts: A pilot study. Inf. Manage. 22 (4), 239–254.
Berner, S., Weber, R., Keller, R.K., 2005. Observations and lessons learned from automated testing. In: Proceedings of the 27th International Conference on Software Engineering. pp. 571–579.
Boehm, B., Abts, C., Chulani, S., 2000. Software development cost estimation approaches—A survey. Ann. Softw. Eng. 10 (1), 177–205.
Box, G.E., Cox, D.R., 1964. An analysis of transformations. J. Royal Stat. Soc.: Ser. B (Methodological) 26 (2), 211–243.
Capgemini, Sogeti, Microfocus, 2020. World Quality Report 2020-21. Technical Report, Capgemini and Sogeti and Microfocus.
Cataldo, M., Nambiar, S., 2009. On the relationship between process maturity and geographic distribution: an empirical analysis of their impact on software quality. In: Proceedings of the 7th Joint Meeting of ESEC/FSE. pp. 101–110.
Claes, M., Mäntylä, M.V., 2020. 20-MAD: 20 Years of issues and commits of mozilla and apache development. In: Proceedings of the 17th International Conference on Mining Software Repositories. pp. 503–507.
Collins, E.F., de Lucena, V.F., 2012. Software test automation practices in agile development environment: An industry experience report. In: 2012 7th International Workshop on Automation of Software Test. AST, IEEE, pp. 57–63.
Corò, F., Verdecchia, R., Cruciani, E., Miranda, B., Bertolino, A., 2020. JTeC: A large collection of java test classes for test code analysis and processing. In: Proceedings of the 17th International Conference on Mining Software Repositories. pp. 578–582.
Cotroneo, D., Pietrantuono, R., Russo, S., Trivedi, K., 2016. How do bugs surface? A comprehensive study on the characteristics of software bugs manifestation. J. Syst. Softw. 113, 27–43.
Croux, C., Dehon, C., 2010. Influence functions of the Spearman and Kendall correlation measures. Stat. Methods Appl. 19 (4), 497–515.
Eldh, S., 2020. Test automation improvement model-TAIM 2.0. In: 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops. ICSTW, IEEE, pp. 334–337.
Eldh, S., Andersson, K., Ermedahl, A., Wiklund, K., 2014. Towards a test automation improvement model (TAIM). In: 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops. IEEE, pp. 337–342.
Fewster, M., Graham, D., 1999. Software Test Automation. Addison-Wesley Reading.
Field, A., Hole, G., 2002. How to Design and Report Experiments. Sage.
Fitzgerald, B., Stol, K.-J., 2017. Continuous software engineering: A roadmap and agenda. J. Syst. Softw. 123, 176–189.
Fowler, M., Foemmel, M., 2006. Continuous Integration, Vol. 122. (14), ThoughtWorks, pp. 1–7, http://Www.thoughtworks.com/ContinuousIntegration.Pdf.
Ganassali, S., 2008. The influence of the design of web survey questionnaires on the quality of responses. In: Survey Research Methods, Vol. 2. (1), pp. 21–32.
García-Mireles, G.A., Moraga, M.A., García, F., Piattini, M., 2015. Approaches to promote product quality within software process improvement initiatives: a mapping study. J. Syst. Softw. 103, 150–166.
Garousi, V., Elberzhager, F., 2017. Test automation: not just for test execution. IEEE Softw. 34 (2), 90–96.
Garousi, V., Mäntylä, M.V., 2016. When and what to automate in software testing? A multi-vocal literature review. Inf. Softw. Technol. 76, 92–117.
Garousi, V., Varma, T., 2010. A replicated survey of software testing practices in the Canadian province of alberta: What has changed from 2004 to 2009? J. Syst. Softw. 83 (11), 2251–2262.
Ghaleb, T.A., Da Costa, D.A., Zou, Y., 2019. An empirical study of the long duration of continuous integration builds. Empir. Softw. Eng. 24 (4), 2102–2139.
Graham, D., Fewster, M., 2012. Experiences of Test Automation: Case Studies of Software Test Automation. Addison-Wesley Professional.
Greer, D., Ruhe, G., 2004. Software release planning: an evolutionary and iterative approach. Inf. Softw. Technol. 46 (4), 243–253.
Groves, R.M., Fowler Jr., F.J., Couper, M.P., Lepkowski, J.M., Singer, E., Tourangeau, R., 2011. Survey Methodology, Vol. 561. John Wiley & Sons.
Hamdan, S., Alramouni, S., 2015. A quality framework for software continuous integration. Proc. Manuf. 3, 2019–2025.
Harter, D.E., Krishnan, M.S., Slaughter, S.A., 2000. Effects of process maturity on quality, cycle time, and effort in software product development. Manage. Sci. 46 (4), 451–466.
Hilton, M., Nelson, N., Tunnell, T., Marinov, D., Dig, D., 2017. Trade-offs in continuous integration: assurance, security, and flexibility. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. pp. 197–207.
Hilton, M., Tunnell, T., Huang, K., Marinov, D., Dig, D., 2016. Usage, costs, and benefits of continuous integration in open-source projects. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 426–437.
Ibrahim, W.M., Bettenburg, N., Adams, B., Hassan, A.E., 2012. On the relationship between comment update practices and software bugs. J. Syst. Softw. 85 (10), 2293–2304.
Int'l Standards Organization, 2001. Iso/IEC 9126-1. URL: https://www.iso.org/standard/22749.html.
Jorgensen, M., 1995. Experience with the accuracy of software maintenance task effort prediction models. IEEE Trans. Softw. Eng. 21 (8), 674–681.
Karvonen, T., Behutiye, W., Oivo, M., Kuvaja, P., 2017. Systematic literature review on the impacts of agile release engineering practices. Inf. Softw. Technol. 86, 87–100.
Kasurinen, J., Taipale, O., Smolander, K., 2010. Software test automation in practice: empirical observations. Adv. Softw. Eng. 2010.
Khomh, F., Dhaliwal, T., Zou, Y., Adams, B., 2012. Do faster releases improve software quality? an empirical case study of mozilla firefox. In: 2012 9th IEEE Working Conference on Mining Software Repositories. MSR, IEEE, pp. 179–188.
Krill, P., 2013. Software engineers spend lots of time not building software. URL: https://www.infoworld.com/article/2613762/software-engineers-spend-lots-of-time-not-building-software.html.
Kumar, D., Mishra, K.K., 2016. The impacts of test automation on software's cost, quality and time to market. Procedia Comput. Sci. 79, 8–15.
Lee, J., Hwang, S., 2012. Software test capability improvement method. In: Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity. Springer, pp. 246–251.
Lin, J.-W., Salehnamadi, N., Malek, S., 2020. Test automation in open-source android apps: A large-scale empirical study. In: 2020 35th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 1078–1089.
Linåker, J., Sulaman, S.M., Maiani de Mello, R., Höst, M., 2015. Guidelines for Conducting Surveys in Software Engineering. [Publisher information missing].
MacCormack, A., Rusnak, J., Baldwin, C.Y., 2006. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. Manage. Sci. 52 (7), 1015–1030.
MarketsandMarkets Research, 2018. Continuous integration tools market. URL: https://www.marketsandmarkets.com/Market-Reports/continuous-integration-tools-market-154327001.html.
McCabe, T.J., 1976. A complexity measure. IEEE Trans. Softw. Eng. (4), 308–320.

Miller, J.N., 1993. Tutorial review—Outliers in experimental data and their treatment. Analyst 118 (5), 455–461.
Pinto, G., Castor, F., Bonifacio, R., Rebouças, M., 2018. Work practices and challenges in continuous integration: A survey with travis CI users. Softw. - Pract. Exp. 48 (12), 2223–2236.
PractiTest, 2020. State of Testing Survey 2020. Technical Report.
Punter, T., Ciolkowski, M., Freimut, B., John, I., 2003. Conducting on-line surveys in software engineering. In: 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.. IEEE, pp. 80–88.
Puri-Jobi, S., 2015. Test automation for NFC ICs using jenkins and nunit. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops. ICSTW, IEEE, pp. 1–4.
Ramler, R., Putschögl, W., Winkler, D., 2014. Automated testing of industrial automation software: practical receipts and lessons learned. In: Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation. pp. 7–16.
Rausch, T., Hummer, W., Leitner, P., Schulte, S., 2017. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In: 2017 IEEE/ACM 14th International Conference on MSR. IEEE, pp. 345–355.
Ruhe, G., Saliu, M.O., 2005. The art and science of software release planning. IEEE Softw. 22 (6), 47–53.
Rwemalika, R., Kintis, M., Papadakis, M., Le Traon, Y., Lorrach, P., 2019. An industrial study on the differences between pre-release and post-release bugs. In: 2019 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 92–102.
Samli, R., Aydın, Z.B.G., Yücel, U.O., 2020. Measurement in software engineering: The importance of software metrics. In: Applications and Approaches to Object-Oriented Software Design: Emerging Research and Opportunities. IGI Global, pp. 166–182.
Sarkar, S., Kak, A.C., Rama, G.M., 2008. Metrics for measuring the quality of modularization of large-scale object-oriented software. IEEE Trans. Softw. Eng. 34 (5), 700–720.
Seaman, C.B., 1999. Qualitative methods in empirical studies of software engineering. IEEE Trans. Softw. Eng. 25 (4), 557–572.
Shahin, M., Babar, M.A., Zhu, L., 2017. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access 5, 3909–3943.
Shapiro, S.S., Wilk, M.B., 1965. An analysis of variance test for normality (complete samples). Biometrika 52 (3/4), 591–611.
Shihab, E., Kamei, Y., Adams, B., Hassan, A.E., 2013. Is lines of code a good measure of effort in effort-aware models? Inf. Softw. Technol. 55 (11), 1981–1993.
Ståhl, D., Bosch, J., 2014. Modeling continuous integration practice differences in industry software development. J. Syst. Softw. 87, 48–59.
Ståhl, D., Mårtensson, T., Bosch, J., 2017. The continuity of continuous integration: Correlations and consequences. J. Syst. Softw. 127, 150–167.
Subramanian, G.H., Jiang, J.J., Klein, G., 2007. Software quality and IS project performance improvements from software development process maturity and IS implementation strategies. J. Syst. Softw. 80 (4), 616–627.
TestSPICE SIG, 2014. TestSPICE - Process Assessment Model. Technical Report, URL: https://i3consult.biz/fileadmin/i3consult/PDF/The_TestSPICE_PAM_3.pdf.
Tosun, A., Bener, A., Turhan, B., 2009. Implementation of a software quality improvement project in an SME: a before and after comparison. In: 2009 35th Euromicro Conference on Software Engineering and Advanced Applications. IEEE, pp. 203–209.
Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., Filkov, V., 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In: Proceedings of the 2015 10th Joint Meeting of ESEC/FSE. pp. 805–816.
Vélez, J.I., Correa, J.C., Marmolejo-Ramos, F., 2015. A new approach to the Box–Cox transformation. Front. Appl. Math. Stat. 1, 12.
Vroon, M., Broekman, B., Koomen, T., van der Aalst, L., 2013. Tmap next: for result-driven testing. Uitgeverij kleine Uil.
Wang, Y., 2018. Test automation maturity assessment. In: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation. ICST, IEEE, pp. 424–425.
Wang, Y., Mäntylä, M., Demeyer, S., Wiklund, K., Eldh, S., Kairi, T., 2020a. Software test automation maturity: A survey of the state of the practice. In: Proceedings of the 15th International Conference on Software Technologies - Vol. 1. ICSOFT, SciTePress, INSTICC, pp. 27–38. http://dx.doi.org/10.5220/0009766800270038.
Wang, Y., Mäntylä, M., Eldh, S., Markkula, J., Wiklund, K., Kairi, T., Raulamo-Jurvanen, P., Haukinen, A., 2019. A self-assessment instrument for assessing test automation maturity. In: Proceedings of the Evaluation and Assessment on Software Engineering. pp. 145–154.
Wang, Y., Mäntylä, M.V., Liu, Z., Markkula, J., Raulamo-jurvanen, P., 2022. Improving test automation maturity: a multivocal literature review. Software Testing, Verification and Reliability e1804. http://dx.doi.org/10.1002/stvr.1804.
Wang, Y., Pyhäjärvi, M., Mäntylä, M.V., 2020b. Test automation process improvement in a DevOps team: Experience report. In: 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops. ICSTW, IEEE, pp. 314–321.
Wendler, R., 2012. The maturity of maturity model research: A systematic mapping study. Inf. Softw. Technol. 54 (12), 1317–1339.
Williams, L., Kudrjavets, G., Nagappan, N., 2009. On the effectiveness of unit test automation at microsoft. In: 2009 20th International Symposium on Software Reliability Engineering. IEEE, pp. 81–89.
Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. pp. 1–10.
Zampetti, F., Vassallo, C., Panichella, S., Canfora, G., Gall, H., Di Penta, M., 2020. An empirical characterization of bad practices in continuous integration. Empir. Softw. Eng. 25 (2), 1095–1135.
Zazworka, N., Shaw, M.A., Shull, F., Seaman, C., 2011. Investigating the impact of design debt on software quality. In: Proceedings of the 2nd Workshop on Managing Technical Debt. pp. 17–23.
Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., Vasilescu, B., 2017. The impact of continuous integration on other software development practices: a large-scale empirical study. In: 2017 32nd IEEE/ACM International Conference on ASE. IEEE, pp. 60–71.

Yuqing Wang received her M.Sc. degree in Software, Systems and Services Development in the Global Environment from the University of Oulu Finland in 2017. She is currently a doctoral candidate who perusing a Ph.D. degree in Information Processing Science at M3S Research Unit of the University of Oulu. Her research interests include empirical software engineering, software test automation, software process improvement, data science, and natural language processing. She has worked on the TESTOMAT project (2017–2020), which gathered 34 academic and industrial partners (from 6 European countries) working on the next level of test automation.

Mika Mäntylä is a professor of Software Engineering at the University of Oulu, Finland. He received a D. Sc. degree in 2009 in Software Engineering from the Aalto University, Finland. His research interests include end of lifecycle software engineering, e.g., software testing, software maintenance, software operations. He has previously worked as an assistant professor at the Aalto University, Finland, and as a post-doc at the Lund University, Sweden. He serves as an associated editor for IEEE Software and Empirical Software engineering. For more information https://mmantyla.github.io/.



Zihao Liu is perusing a Ph.D. degree in Information Processing Science at M3S Research Unit of the University of Oulu. He finished his M.Sc. degree in Software, Systems and Services Development in the Global Environment from the University of Oulu Finland in 2018. His research interests contain empirical software engineering, software process improvement, Artificial intelligence (AI), Internet of Things (IoT), and Virtual Reality.

Jouni Markkula is a Professor in Software Engineering and Vice Head of Empirical Software Engineering research unit M3S at the University of Oulu. He holds a Ph.D. in Computer Science, Lic.Ph. in Statistics and M.Sc. in Statistics and Philosophy. His main research interests include empirical software engineering, research methods, knowledge management, decision making, data intensive service design and software engineering education. Before the University of Oulu, Prof. Markkula has been working at the Information Technology Research Institute of the University of Jyväskylä as a Research Director. He has published 100+ international peer-reviewed journal and conference articles and book chapters.