How the Experience of Development Teams Relates to Assertion Density of Test Classes

Gemma Catolino[1], Fabio Palomba[2], Andy Zaidman[3], Filomena Ferrucci[1]
[1]University of Salerno, Italy, [2]University of Zurich, Switzerland, [3]Delft University of Technology, The Netherlands
gcatolino@unisa.it, palomba@ifi.uzh.ch, a.e.zaidman@tudelft.nl, fferrucci@unisa.it

***Abstract*—The impact of developers' experience on several development practices has been widely investigated in the past. One of the most promising research fields is software testing, as many researchers found significant correlations between developers' experience and testing effectiveness. In this paper, we aim at further studying this relation, by focusing on how development teams' experience is associated with the assertion density, *i.e.,* the number of assertions per test class KLOC, that has previously been shown as an effective way to decrease fault density. We perform a mixed-methods empirical study. First, we devise a statistical model relating development teams' experience and other control factors to the assertion density of test classes belonging to 12 software projects. This model enables us to investigate whether experience comes out as a statistically significant factor to explain assertion density. Second, we contrast the statistical findings with a survey study conducted with 57 developers, who were asked their opinions on how developer's experience is related to the way they add assertions in test code. Our findings suggest the existence of a relationship: on the one hand, the development team's experience is a statistically significant factor in most of the systems that we have investigated; on the other hand, developers confirm the importance of experience and team composition for the effective testing of production code.**

***Index Terms*—Assertion Density; Developers' Experience; Mixed-Methods Empirical Study.**

I. Introduction

It is often said that "failure leads to success". One has to have experienced failure to truly understand what it takes to succeed. This is true in many fields, and perhaps also in the area of software testing. Pham *et al.* [1] have already alluded to it, when they conjectured that junior developers do not see the need to write test cases, or at least, not that many test cases; their reasoning being that junior developers have likely not experienced the consequences of inadequate testing [1].

The research community has widely investigated developers' experience as a factor in different contexts related to software maintenance and testing [1]–[10]. In particular, several studies focused on the correlation between developers' experience and the effectiveness of testing activities [1], [4]–[7], finding a high correlation between the two phenomena. Following these previous achievements, in this paper we aim at further understanding the relation between developers' experience and software testing practices, which are fundamental for program comprehension, understandability, and maintainability.

Among the different ways to measure test code effectiveness (*e.g.,* branch or mutation coverage [11]–[13]), we focus on test assertions, which were originally described by Alan Turing [14] as an effective way to prove the correctness of a program and later shown as actually useful to improve testing effectiveness. Specifically, researchers found that a high *assertion density*, *i.e.,* the number of assertions per test class KLOC, is associated with a decrease of faults in production code [15] and makes software systems more stable over their evolution [16], as the assertions verify the internal state of a program at runtime [17].

In the context of this paper, we conjecture that *the experience of development teams in charge of developing a test class is correlated to its assertion density*. In other words, teams composed of more expert developers add more assertions to better verify the production code. Should this conjecture be confirmed, the experience factor would represent an instrument based on which project managers can compose testing teams that are more effective and less likely to miss faults in production code. To test our conjecture, we perform a mixed-methods empirical study [18] that aims at addressing the following two main research questions:

- **RQ$_1$.** *To what extent is the experience of development teams correlated to the assertion density?*
- **RQ$_2$.** *How do developers perceive experience as a relevant factor for assertion density?*

In the first place, we collect data of 12 open-source projects and build a statistical model relating development teams experience (and other control factors) to assertion density. In addition, we contrast the statistical results with a survey study featuring the opinion of 57 developers, who were asked about the relation between experience and assertions.

The results of both the studies converge toward a clear conclusion: the experience of development team is a statistically significant factor to explain assertion density; this is also reflected in the opinions of developers, who explicitly state the importance of testing teams' composition and experience for effective testing of production code.

Based on our findings, we identify two main implications for both the research community and practitioners.

On testing team composition. The findings reported in the paper show that taking into account the experience of development teams might have an impact on assertion density, meaning that practitioners and project managers should consider this aspect when allocating testing tasks. At the same time, the research community is called upon to investigate novel experience-aware methodologies and techniques that enable/optimize the allocation of resources.

On human-oriented testing. As a side effect, the results of our study highlight the importance of considering human aspects in software testing and how they can impact source code quality. On the one hand, this confirms recent findings in the field of software maintenance and evolution [19]–[22] as well as the correlations observed between developer's experience and testing practices [4]–[7]; on the other hand, researchers in the field of software testing should further explore these aspects and understand how to measure them and the extent to which good people management can lead to better software quality and reliability.

Structure of the paper. Section II overviews the literature related to previously studied uses of developers' experience, and assertions. In Section III we discuss the research methodology adopted to build the statistical model, while in Section IV we report the results of the study. Section V examines the threats to the validity of the study and the way we mitigated them. Finally, Section VI concludes the paper and provides insights on our future research agenda.

II. Related Work

As our work is at the intersection between the use of assertions and developers' experience, due to space limitations, we only report the main work related to these two aspects.

A. On assertion density

Rosenblum [23] defined assertions as *"formal constraints on software system behavior that are commonly written as annotations of a source text. The primary goal in writing assertions is to specify what a system is supposed to do rather than how it is to do it"* [23]. Based on this definition, the use of asserts has been both suggested and investigated by many researchers in different contexts [15], [24]–[28].

In particular, researchers constantly advised practitioners to introduce assertions when testing software systems in order to promote the automatic checking for program failures [26]. Chen *et al.* [28] found that assertion density, *i.e.,* the number of assertions per KLOC, is strongly correlated to coverage-based test-suite reduction. Estler *et al.* [16] investigated the use of pre- and post-conditions in 21 object-oriented projects, finding that program elements including contracts/asserts tend to be more stable over time. More importantly, Kudrjavets *et al.* [15] showed that bug density decreases when the assertion density increases, meaning that the higher the number of assertions, the lower the number of bugs in production code. Finally, there are some works very close to ours [29]–[31]. In particular, Casalnuovo *et al.* [31] collected asserts in C and C++ programs in order to assess a relationship between asserts and defect occurrence. As a results they found that methods with asserts do have significantly fewer defects. They replicated this analysis in a subsequent study [30] showing also that developers with higher ownership and experience are more likely to add assertions. Finally, Kochar *et al.* [29] analyzed the correlation between the use of assertions and the presence of defects, but also whether and how developer-related factors are correlated to the addition of assertions. In particular, they show that there exists a significant relationship between assertions and the presence of defects; also, developers' ownership leads to more assertions being added in test code.

To summarize, the aforementioned investigations indicate that having a high assertion density represents a key factor influencing quality aspects of test cases. Our work builds upon these previous findings and aims at further investigating whether developers' experience might play a role in the way developers introduce assertions when testing software systems.

B. On the use of developers' experience

Recent findings have shown how developers' experience constitutes a key factor to carefully consider during maintenance tasks [2], [3], [8]–[10], [32], [33]. For instance, Bhatt *et al.* [10] found that human and organizational factors such as organization climate, customer attitude, and developers' experience have a significant influence on software maintenance effort. Similarly, Jørgensen *et al.* [8] reported that more expert developers promptly deal with the complexity of maintenance tasks. Finally, Li *et al.* [9] discovered that maintainers' experience, tool support and domain knowledge are the most influential cost drivers of bug fixing operations.

Besides software maintenance, several studies have pointed out the role of experience when performing testing activities [1], [4]–[7]. Specifically, Kanij *et al.* [5] investigated which factors influence testing effectiveness through a user study. The results show that tools and training, but also human-centered factors like personality characteristics and experience are those more related to the ability of discovering bugs in production. The same authors [6] also studied the correlations between personality characteristics of developers and testing effectiveness, confirming the central role of experience. Rooksby *et al.* [7] reported about the collaborative aspects of testing activities, while Pham *et al.* [1] interviewed 97 computer science students, exploring their experience and attitudes regarding testing. A key result of these studies is that novice developers seemingly have less understanding of what should be tested. Finally, Beer and Ramler [4] investigated the role of developers' experience during testing activities in three projects at Siemens. The results showed how test case design is based on experience in all three projects and that experience-based testing is an important supplementary approach to requirements-based testing.

Our work can be seen as complementary to those discussed above. Indeed, while previous work clearly pointed out the role of developers' experience for testing activities, we aim at further understanding this relation, by considering whether and how the experience of developers that touch a test class (*i.e.,* development teams) can impact the assertion density, building a statistical model, and corroborating the results conducting a survey analysis involving a large number of developers.

III. Research Methodology

This section reports the methodology that we have followed to study the relation between the experience of development teams and assertion density.

A. Hypothesis and Research Questions

The *goal* of the empirical study is to investigate the relationship between the experience of development teams and the assertion density, with the *purpose* of understanding the extent to which team experience can have a correlation with the number of assertions present in a test class. The *quality focus* is on the relation between team experience and assertion density, while the *perspective* is that of project managers who want to understand how much team experience can be a factor to consider when allocating resources for testing. More specifically, our study was driven by the following hypothesis:

H_0 - The experience of development teams correlates with the assertion density of test classes.

In other words, we believe that teams composed of more expert developers write more effective test cases. It is important to note that we intentionally focused on assertion density because previous research has shown its relation to test code effectiveness [15]–[17]. As such, we are interested in determining which factors are correlated to assertion density and how development team's experience contribute to it. We are aware that other factors may relate to the test code effectiveness (*e.g.,* branch or mutation coverage [11]–[13]), but an analysis of the factors influencing them is out of the scope of this paper. The aforementioned hypothesis has led to the definition of two main research questions:

- **RQ_1.** *To what extent is the experience of development teams correlated to the assertion density?*
- **RQ_2.** *How do developers perceive experience as a relevant factor for assertion density?*

As detailed in the next subsections, we addressed our research questions by means of a mixed-methods approach [18]. In **RQ_1**, we built a statistical model relating experience of development teams to the assertion density of test classes, while in **RQ_2** we conducted a survey study with 57 developers.

B. *RQ_1 — Research Methodology*

In this section, we overview the methodology adopted to address our first research question.

Context Selection. The *context* of the study consists of the 12 open source systems whose characteristics are shown in Table I. Starting from the list of projects available on GITHUB,[1] we first ordered them based on the number of tests and excluded those which had less than 100 JUnit test classes. Then, we re-ordered the list based on the repository activity, *i.e.,* the number of commits performed over the last year. With these two criteria, we randomly selected 12 systems having a high number of tests and being actively developed: the first criterion was used to set a minimum number of test classes the considered systems should have to be part of our study, *i.e.,* no statistically sound method can be applied with few data points; the latter was an important requirement aimed at reducing possible threats to construct validity in the way development team's experience is computed (see Section V for further discussion). It is also worth noting that the randomly selected systems come from different application domains and have different characteristics, especially in terms of number of developers and number of classes: as such, this selection process mitigates possible threats to external validity. To enable the replication of our study, we made the dataset available in our online appendix [34].

[1]https://github.com

Table I: Characteristics of the Software Projects in Our Dataset

System	# Assertions	#Commits	#Dev.	#Classes	#Methods	KLOCs
Adempiere	421	14,131	20	4,922	104,866	1,112
Camel	45,035	35,143	447	17,171	111,195	940
Commons-Lang	14,517	5,363	110	308	6,292	97
Groovy	1,914	15,578	267	1,453	19,198	207
Closure	14,986	13,568	406	1,182	27,867	268
Eclipse Che	5,525	7,550	113	7,871	44,678	357
Guava	38,049	4,861	171	3,057	46,964	480
Jabref	4,642	12,262	184	1,429	9,874	86
Metasfresh	8,641	23,561	14	11,433	152,367	1,407
Mockito	2,555	4,946	138	816	4,783	45
RxJava	17,041	5,515	236	1,616	25,211	266
xWiki	10,096	36,182	94	4,477	26,901	324
Overall	**163,422**	**178,660**	**2,200**	**55,735**	**580,196**	**5,589**

Extracting Development Teams Composition. The first step needed to answer our research question was the actual identification of development teams in our dataset. To this aim, we followed the same definition of development team used by many previous studies (*e.g.,* [35], [36]): in particular, a team is defined as the set of developers who have added/modified/removed lines of code to a certain (test) class. We are aware that such a definition might lead to the approximation of the real composition of the development teams of the considered projects, *e.g.,* a team member might not necessarily contribute to the development of the test code. However, this identification strategy is the only one available so far and, according to previous studies [35], represents a valid heuristic to estimate both size and composition of open-source development teams.

Building a Statistical Model. Once we have extracted the development teams composing our subject projects, we proceeded with the definition of a statistical model relating development team experience to assertion density. In the following, we overview the steps that we have taken.

Response Variable Definition. The response variable in our statistical model is the *assertion density*, which has been defined by Kudrjavets *et al.* [15] as follow:

$$assertion\ density(T_i) = \frac{\#assertions_{T_i}}{KLOC_{T_i}} \quad (1)$$

where *#assertions*$_{T_i}$ represents the total number of assertions in a test class T_i and *KLOC*$_{T_i}$ is the thousands (kilo) of lines of code of T_i.

Independent Variables Definition. Our goal was to measure the extent to which the experience of development teams is correlated to the assertion density of test classes. Thus, our independent variables were represented by metrics computing team experience under different perspectives. We relied on the metrics defined by Kamei *et al.* [37]: experience (EXP), recent experience (REXP), and subsystem experience

(SEXP). The rationale behind their selection is twofold: on the one hand, they can measure experience in three orthogonal dimensions [37], [38], thus allowing us to more comprehensively verify our research hypothesis. On the other hand, these metrics have been widely employed in the past by the research community and are well-established in the field [39], [40]. Given a developer d, the first metric computes the total number of commits performed by d on a certain test class T_i (hereafter, we name this metric as "T-EXP" to highlight that it refers to test classes); REXP refers to the number of commits performed by d on T_i over the last three months ("T-REXP"), while SEXP computes the number of commits performed by d on the package containing T_i ("T-SEXP"). Note that while the last metric does not directly consider T_i, it still makes sense because developers that are in charge of testing an entire subsystem might have more confidence with the test classes it contains.

As we consider the overall experience of a development team, we needed to compute these metrics at team-level. To do so, we have followed a similar process to that of previous work [35], [36]. Specifically, for each test class T_i:

- We identified the development team corresponding to T_i (as described before in this section);
- For each developer, we computed T-EXP, T-REXP, and T-SEXP;
- We aggregated the values of single developers using the median operator, *i.e.,* the final development team experience was given by the median experience of the involved developers. It is important to note that we have adopted the median to mitigate the influence of possible outliers (*e.g.,* a team member having much more experience than another one): nevertheless, this choice did not bias our conclusions, as similar results have been observed when considering the average experience of developers in a team (see Section V for more details).

Besides computing the experience of development teams on test classes, we also measured how the developers belonging to those teams are expert globally, *i.e.,* independently from test classes. Indeed, it may be that the overall experience of a developer influences her actions on source code, rather than the specific experience on tests. To account for this aspect, we computed EXP, REXP, and SEXP of development teams on all classes of the considered systems, *i.e.,* without considering tests only. For the sake of readability, in the following we name them as O(verall)-EXP, O-REXP, O-SEXP. It is worth noting that other experience metrics have been proposed in literature, like the commit-tenure [20], [36], [41] which computes the experience of developers looking at the contributions done over all GITHUB projects. However, these metrics are domain-agnostic, meaning that they do not distinguish the type of projects a developer contributes to (*e.g.,* third-party library or Android app) and, therefore, can fail in identifying the actual experience of a developer in a specific project. As we were interested in assessing how developers add assertions in the specific context of the considered projects, we preferred to compute within-project metrics such as EXP, REXP, and SEXP. A further investigation of domain-agnostic experience metrics is part of our future work.

Control Variables. Although we conjectured that development team experience is correlated to the assertion density of test classes, it is worth remarking that other factors related to both the structure of production and test code (*e.g.,* number of lines of code) might represent an important source of information to understand the response variable [42]–[44]. To account for this aspect, we have defined a list of technical factors having the role to control possible confounding factors when evaluating the role of development team experience. More specifically, we have computed the four categories of metrics described in the following:

Size. The first factor we have considered is the size of both production and test classes. Indeed, it might be possible that the number of assertions present in a test class is simply a reflection of the number of lines of code or the number of methods belonging to the test class (*i.e.,* the larger the test class, the higher the number of assertions) or the tested class (*i.e.,* the test requires more assertions to test a large production class); thus, we considered the metrics Number of Methods (NOM) [45]—computed for both production and test classes, *i.e., prod.class.nom* and *test.nom*—and Lines of Code (LOC)—only for production code, *i.e., prod.class.loc*—as control factors in our model;

Complexity. Intuitively, production classes having a higher complexity require more effort to be tested [46], [47]. Following this conjecture, we computed the Weighted Methods per Class (WMC) metric [48], which measures the degree of complexity of a production class, *i.e., prod.class.wmc*. It is important to remark that WCM uses the McCabe Cyclomatic Complexity metric [49], which in turn is known to be important with respect to the number of test cases (or assertions) that need to be written [50]. Similarly, we also computed WMC on test classes to check the extent to which the complexity of the test is correlated to the assertion density, *i.e., test.wmc*;

Cohesion. Low cohesive classes might contain code responsible for more than one responsibility [48]. As a consequence, the corresponding test classes might require more assertions to verify the behavior of the production code. To control for this aspect, we computed the Lack of Cohesion of Methods (LCOM5) defined by Henderson-Sellers *et al.* [51] on the production classes of our dataset, *i.e., prod.class.lcom*. Also in this case, we computed LCOM5 on test classes too: indeed, low cohesion of tests has been shown to be an important factor for test effectiveness [52], [53], *i.e., test.lcom*;

Coupling. Production classes having a high level of coupling tend to be less maintainable [42], [54]. As a result, the corresponding test classes might require more assert statements in order to carefully verify the behavior of the tested code. For this reason, we compute the Coupling

Table II: Complete list of survey questions.

n.	Question	Evaluation Criteria
Section I. Background		
1	What is your current job?	Multiple Choice (with the possibility, in addition, to write the job)
2	Experience in: **2.1 -** Programming **2.2 -** Industrial development **2.3 -** Verification/Testing of Programs	Multiple Choice (*e.g.,* No Experience, 1-3 Years, More than 5 years)
3	What is your company size?	Multiple Choice (*e.g.,* more than 250 employees)
4	What is your team size?	Multiple Choice (*e.g.,* 5-10 team, Just me)
Section II. Relevant factors when writing assertions		
5	Please rate the importance of the following aspects for writing assertions: **5.1 -** Your testing skills **5.2 -** Your experience within the domain of the project (*e.g.,* in past you worked on similar code and know where to check for bugs) **5.3 -** Your experience within the project (*i.e.,* the knowledge accumulated on the project) **5.4 -** Authority of the developer who wrote the production code (*e.g.,* For example, whether the production code has been developed by a core developer or a developer whose reputation is high, but he/she doesn't test its code) **5.5 -** The characteristics of the production code that should be tested (*e.g.,* if you have to test a long or complex method/class) **5.6 -** The characteristics of the test code (*e.g.,* if it already has assertions) **5.7 -** The presence of test smells/anti-patterns (*e.g.,* a test method that exercises more than one production method)	Likert Scale (*i.e.,* from *Not at all important* to *Extremely important*)
	5.8 - Other	Text box
Section III. Further opinions		
6	Can you please provide a brief explanation to your answers, by reporting how do you decide to add an assertion in your tests and why these factors are (not) important?	Open Question

Between Object Classes (CBO) [48] on the considered production classes, *i.e., prod.class.cbo*. Moreover, we compute CBO on test units because coupling between tests can affect test effectiveness [52], *i.e., test.cbo*.

In the end, the statistical model comprises a total of nine control factors and six independent variables. For each of them, we initially compute the standard deviation in order to understand the distribution of each factor and whether there are outliers that might possibly create bias in the statistical model [55]. Based on the observed distributions, we then decide to use the natural logarithm of the computed metrics as independent variables of the model—the logarithm correction is recommended to reduce the impact of outliers on the results of statistical models [55].

It is important to point out that an intuitively interesting control factor could be *test coverage* [56], [57], a measure that expresses how many lines, branches or methods of production code are actually exercised by the tests. However, the coverage measure typically does not represent the *quality* of a test: a more high-level (system/integration) test might cover production code up to a certain percentage, while more fine-grained (unit) tests might cover that same code up to the same percentage of coverage [47]. Yet, more fine-grained tests are likely to be more helpful to the developer when trying to locate a defect. Because of the potentially "misleading" nature of test coverage, we have decided not to consider it as a factor; we aim to further investigate this aspect in our future research agenda.

Statistical Modeling. Once we have collected all the required information, we devise a generalized linear model (GLM) [58] relating development team experience and control factors to assertion density for each of the projects in our dataset. This statistical technique is used to fit a function describing the continuous response variable (the assertion density in our case) relying on a set of categorical and/or continuous variables (in our case, experience of development team and further control factors). We have used this statistical modeling approach for two reasons. On the one hand, it is able to analyze the simultaneous effects of both independent variables and control factors on the response variable [59]. On the other hand, it does not assume the underlying distribution of data to be normal: in our case, we have verified the normality of the distribution exploiting the Shapiro-Wilk test [60], which fails to reject the null-hypothesis, *i.e.,* our data is not normally distributed and, as such, we had to rely on GLM [58]. Note that we also verified that the other assumptions made by the statistical method (*e.g.,* errors are independent but not normally distributed) are valid adopting the standard diagnostics tools provided by the `regdiag` package[2] available in `R`.[3]

More formally, let $Logit(\pi_t)$ be the explained proportion of assertions in a test t, let β_0 be the log odds of the assertion density being increased in a test, and let the parameters $\beta_1 \cdot t - exp_t$, $\beta_2 \cdot t - rexp_t$, $\beta_3 \cdot t - sexp_t$, $\beta_4 \cdot loc_t$, etc. be the differentials in the log odds of being the assertion density increased for a test with characteristics $t{-}exp_{t-mean}$, $t - rexp_{t-mean}$, $t - sexp_{j-mean}$, etc., the devised statistical model is represented by the function:

$$\begin{aligned} Logit(\pi_t) &= \beta_0 + \beta_1 \cdot t - exp_t + \beta_2 \cdot t - rexp_t + \\ &+\beta_3 \cdot t - sexp_t + \beta_4 \cdot loc_t + \\ &+...(other\ vars\ and\ \beta\ omitted\ for\ space\ reasons) \end{aligned} \quad (2)$$

To implement the model, we rely on the `glm` function available in the `R` toolkit. To avoid multi-collinearity we use the `vif` (Variance Inflation Factors) function [61] implemented in `R` to discard non-relevant variables, putting a threshold value equal to 5 as recommended in literature [61]. This method provides an index for each independent variable that measures how much the variance of an estimated regression coefficient is increased because of collinearity. The square root of the variance inflation factor indicates how much larger the standard error is, compared to what it would be if that variable were uncorrelated with the other predictor variables in the model. Based on this information, we can understand which metric produces the largest standard error, thus allowing the identification (and removal) of the metric that is better to drop from the model.

[2]https://goo.gl/Z9WRrr

[3]https://www.r-project.org

Data Analysis. Once we have built the statistical model, we address $\mathbf{RQ}_1$ by assessing whether the coefficients assigned by the statistical model to the independent variables are statistically significant ($p < 0.05$). In other words, we verify that, despite the presence of the control factors, the development team's experience represents an important (*i.e.,* statistically significant) factor when explaining the assertion density of test classes. Moreover, in order to measure the goodness of fit of our model we computed the *R-square* coefficient [62], which is a measure determining how well the model fits our data and how well it predicts new unseen observations [62]. The coefficient is implemented in the `rsq` package available in `R`.

C. $\mathbf{RQ}_2$ — Research Methodology

In $\mathbf{RQ}_2$, we aim at triangulating the results that we achieve in the previous research question and understand developers' opinions on the relation between experience and assertion density. For this purpose, we conduct a survey study, whose details are reported in the following.

Survey design. We define an anonymous questionnaire composed of three main sections. In many cases, making a survey anonymous leads to more honest feedback [63]; indeed, previous studies have shown that when respondents are aware that they will not be tied to their answers, a researcher may get more insights [63], [64]. The structure of the survey, along with the expected response type, is reported in Table II.

In the first section we collect demographic information of the participants, including programming/testing experience as well as some information about the size of company and her/his team: we use this information to characterize the sample of developers taking part in the study. In the second section, we inquire participants about their opinions on the assertion mechanism with the aim of gathering insights that can address our research question. In particular, we ask what are the crucial aspects for writing assertions: we provide them with a predefined list of factors potentially being relevant when writing assertions (*e.g., 'Testing Skills'* or *'Experience in domain of the project'*) and requested them to rate the importance of each of them using a 5-points Likert scale [65] ranging from *'Not at all important'* to *'Extremely important'*. Of course, participants could also indicate additional entries by filling out a text box with other relevant aspects. It is important to note that in this section we explicitly ask about the factor that we want to measure, *i.e.,* developer's experience. With the answers provided to the experience-related questions, we could assess how much the experience counts for developers and possibly corroborate the results of $\mathbf{RQ}_1$. Furthermore, in the last part of this survey we asked participants to give further opinions on whether and how tester's experience is related to the addition and management of assertions. In doing so, we aimed at collecting additional insights that can help us in answering $\mathbf{RQ}_2$ and better understanding developers' opinions on the factors related to the assertion density.

Survey dissemination. We have created the questionnaire using a GOOGLE survey module and made it accessible from December 10th, 2018 to January 10th, 2019. We have first advertised it using our personal social network accounts, *i.e.,* the survey was available on FACEBOOK, TWITTER, and LINKEDIN. Then, we have posted it on REDDIT,[4] targeting three specific sub-groups such as SOFTWARETESTING,[5] COMPSCI,[6] and LEARN JAVA.[7] We have selected these sub-reddits because (i) they allow for the advertisement of surveys, (ii) they have a large amount of subscribers (≈35,000, overall), and (iii) they are all directly related to Java and/or JUnit. Finally, we have also contacted developers through our personal contacts.

Survey recruitment. To stimulate developers to participate in the study, we have followed the guidelines provided by Flanigan *et al.* [66]. As such, we mitigate common issues possibly arising in survey studies and affecting the response rate by keeping the survey short, respecting the anonymity of participants, and preventing our influence in the answers. As a result, we have collected 57 fully compiled questionnaires: 4 of them came from personal contacts, 5 from TWITTER, and the remaining 48 from the REDDIT sub-groups.

IV. ANALYSIS OF THE RESULTS

In this section we report on the analysis of the results obtained for the two research questions formulated in our study. For $\mathbf{RQ}_1$ we show the results of the statistical models built for each system considered in the study. Subsequently, for $\mathbf{RQ}_2$, we show through charts and box plots the results of our survey. The raw data and fine-grained overview of the results of the study are available in the online appendix [34].

A. $\mathbf{RQ}_1$ - Statistical Models results

Table III shows the results of the 12 models we have built, one for each project in our dataset. Specifically, the set of tables contain, for each independent variable and control factor investigated (column "Factor"), the value of the estimate in the regression model (column "Est.") and the standard error (column "S.E."). The statistical significance is given by the number of stars, *i.e.,* '***' indicates a $p < 0.001$, '**' a $p < 0.01$, '*' a $p < 0.05$, and '.' a $p < 0.1$. It is important to note that when a certain metric is associated with a blank cell in the tables, it indicates that the metric has been excluded by the model as a result of the `vif` analysis [61]. For the sake of space limitation, detailed results *i.e.,* goodness of fit for each model as well as the script used in order to compute these analyses are available in our online appendix [34].

Looking at the tables, it seems that the models follow a common pattern: indeed, metrics related to the size and complexity of test and production classes, *i.e.,* NOM and WMC, are usually discarded or have limited statistical significance in only few cases. In particular, the result of the WMC metric is pretty surprising: it measures the complexity of production code and directly impacts testing efforts [49], however it turned out that it has often a limited power. In other words, most

[4]https://www.reddit.com/
[5]Link omitted for double-blind rules
[6]Link omitted for double-blind rules
[7]Link omitted for double-blind rules

	Adempiere			Camel			Common-Lang		
	Est.	S.E.	Sig.	Est.	S.E.	Sig.	Est.	S.E.	Sig.
(Intercept)	-2.594	1.176	*	-3.966	0.391	***	-2.498	1.185	*
o.exp	-1.042	0.765		0.013	0.19		0.137	0.619	
o.sexp	-0.849	0.716		-0.304	0.142	*	-0.115	0.49	
o.rexp	-0.625	0.371	.	-0.014	0.154		0.572	0.518	
t.exp	-0.763	1.01		0.017	0.330		0.011	1.114	
t.sexp	0.526	2.342		0.018	0.329		-0.849	1.069	
t.rexp	3.452	1.032	**	3.566	0.487	***	5.6	0.987	***
prod.class.loc				0.023	0.037		-0.105	0.114	
prod.class.cbo	-0.003	0.006		-0.005	0.006		-0.003	0.02	
prod.class.lcom				0.014	0.012		-0.005	0.04	
prod.class.nom							0.005	0.009	
prod.class.wmc							0.002	0.016	
test.nom									
test.cbo	-0.013	0.02		0.014	0.005	*	-0.013	0.02	
test.lcom	0.001	0.002		0.001	0.0004	***			
test.wmc									

	Groovy			Closure			Eclipse Che		
	Est.	S.E.	Sig.	Est.	S.E.	Sig.	Est.	S.E.	Sig.
(Intercept)	-5.525	1.624	***	-6.049	0.772	***	-5.819	0.447	***
o.exp	0.23	0.845		0.279	0.51		0.402	0.374	
o.sexp	-0.895	0.608		0.531	0.414		-0.224	0.312	
o.rexp	-0.49	0.683		-0.266	0.287		-0.168	0.204	
t.exp	-0.947	1.471		-0.913	1.375		-1.324	0.395	***
t.sexp	-0.187	1.494		1.784	0.435	***	3.926	0.865	***
t.rexp	5.416	1.685	**	6.668	0.386	***	4.183	0.358	***
prod.class.loc	0.25	0.151	.	-0.033	0.063		0.326	0.051	***
prod.class.cbo	0.011	0.027		-0.002	0.006		-0.025	0.005	***
prod.class.lcom	-0.003	0.0485					0.0001	0.0003	
prod.class.nom	0.002	0.011							
prod.class.wmc	0.013	0.023							
test.nom									
test.cbo	0.017	0.026		0.041	0.009	***	-0.005	0.004	
test.lcom							0.0007	0.0002	***
test.wmc									

	Guava			Jabref			Metasfresh		
	Est.	S.E.	Sig.	Est.	S.E.	Sig.	Est.	S.E.	Sig.
(Intercept)	-4.333	0.255	***	-4.019	0.866	***	-7.094	0.558	***
o.exp	0.523	0.200	**	0.272	0.345		0.837	0.353	*
o.sexp	0.643	0.166	***	0.343	0.29		0.323	0.309	
o.rexp	-0.211	0.116	.	-0.573	0.203	***	0.17	0.2	
t.exp	1.155	0.227	***	-0.166	1.471		3.944	0.576	***
t.sexp	1.613	0.527	**	0.168	0.92		0.595	0.904	
t.rexp	2.482	0.268	***	5.189	0.524	***	3.807	0.721	***
prod.class.loc				0.21	0.041	***	0.164	0.047	***
prod.class.cbo	0.010	0.003	***	-0.020	0.006	***	-0.009	0.003	**
prod.class.lcom				-0.0002	0.0002				
prod.class.nom									
prod.class.wmc									
test.nom				0.032	0.006	***			
test.cbo	-0.003	0.003		-0.033	0.008	***	-0.001	0.004	
test.lcom				-0.0005	0.0003	*	-0.0003	0.0006	
test.wmc	0.044	0.02	*	-0.211	0.053	***			

	Mockito			RxJava			xWiki		
	Est.	S.E.	Sig.	Est.	S.E.	Sig.	Est.	S.E.	Sig.
(Intercept)	-0.566	1.430		-3.240	0.626	***	-4.047	2.78	***
o.exp	1.149	0.526	*	-0.168	0.242		-6.953	2.193	**
o.sexp	0.593	0.430		0.521	0.205	*	2.725	1.835	
o.rexp	-0.459	0.321		-0.630	0.146	***	-4.956	1.226	***
t.exp	-4.353	2.313	.	1.633	1.02		4.330	3.471	***
t.sexp	-4.035	1.489	**	-0.768	0.65		-8.115	5.590	
t.rexp	3.806	0.887	***	1.38	0.467	**	8.816	3.782	*
prod.class.loc	0.073	0.088		0.243	0.045	***	6.157	1.984	**
prod.class.cbo	-0.037	0.015	*	-0.051	0.013	***			
prod.class.lcom	-0.0003	0.0004							
prod.class.nom									
prod.class.wmc									
test.nom									
test.cbo	-0.006	0.012		0.026	0.009	**	-1.710	2.305	***
test.lcom	0.0007	0.0005					-4.525	8.421	
test.wmc	-0.056	0.083		-0.250	0.031	**			

Table III: Results achieved by the statistical models - S.E. = Standard Error, Sig. = Statistical significance

of the metrics that are supposed to be correlated to the way developers test production code in terms of assertion density are instead not at all or not very significant.

With respect to the role of development team experience, as shown in the tables, at least two of the experience-related variables are significant in all the projects when describing the phenomenon of assertion density; in particular, O-EXP is statistically significant in four projects, while O-SEXP and O-REXP are statistically significant in three and five projects, respectively. The statistical relevance increases when considering the metrics specifically related to testing experience: indeed, T-EXP and T-SEXP are statistically significant factors in five and four projects, respectively, while T-REXP is the only metric that is significant in all the systems considered.

Thus, our findings seem to confirm our hypothesis: *the development team's experience is correlated to the assertion density of test classes*. As a side effect, we can confirm what previous works reported on the relation between experience and testing practices, namely: the most expert developers tend to better test production code [1], [4]–[7]. Furthermore, our findings further stimulate and suggest the need for more research on how developer-related factors can be employed within software testing techniques.

Besides development teams experience, we also observe that other control variables, mainly related to cohesion and coupling, can partially explain the phenomenon of interest. As an example, let us consider the case of ECLIPSE CHE. Here all the metrics related to testing experience, along with the test LCOM and production CBO are significant factors explaining the assertions density of the system. On the one hand, this means that *test cohesion and product coupling can represent important elements to assess the dependent variable* (this is true for more than one system considered); on the other hand, it is worth noting that this is the third largest system considered in our study (see Table I), possibly indicating that the assertion density in larger systems tends to be mainly explained by test class-related factors.

If we consider the other systems, we find that in GUAVA all the metrics related to the developers' experience are significant. Analyzing the project deeper, we find that the statistical significance of O-SEPX and T-SEXP can be explained by the solid package division and good distribution of classes within the system: this is confirmed by the value of the Modularization Quality (MQ) metric [67] of the project, that is 0.85. Such a good modularization likely made developers more focused on specific aspects of the system and, as a consequence, more expert of the way they should be exercised. A similar discussion can be held when considering the METAFRESH project, where we have found that O-EXP and T-EXP are significant. A possible explanation could be given by looking at its repository. In particular, we observe that only 4 out of 14 contributors have continuously been active in the project right from the start and have developed all the tests. As such, the testing experience is concentrated within a small group of developers; this aspect is correctly identified by both metrics, that are, therefore, significant in this case. There are also smaller systems like ADEMPIERE, GROOVY and APACHE COMMON LANG, where we observe that the recent experience on test classes (T-REXP) is relevant for explaining assertion density. Looking at the ADEMPIERE repository, we notice that over the last three months there is an increasing number of commits and the development of both production and test code was very active: likely, this has contributed to the statistical relevance of T-REXP and O-REXP. As for APACHE GROOVY, we notice

that the number of commits is quite low, so T-REXP could better explain the phenomenon of assertion density. Finally, for APACHE COMMON LANG, the number of developers working on test cases is pretty low (20 out of 113 contributors), and they frequently modify tests to make them compliant with the changes of the production code: therefore, T-REXP turns to be the metric that correlates better with assertion density.

As mentioned in Section III-B, we also verify the goodness of fit of the devised statistical model using the *R-square* coefficient, that is a metric to determine how well the model fits our data and how well it predicts new unseen observations [62]. The average value of the coefficient is 0.25. According to previous studies [68], [69], this value is considered "moderate", meaning that the statistical models we have built can fit the data reasonably well and, therefore, can well explain the phenomenon of interest, *i.e.,* assertion density. The detailed results achieved for each project are available in the appendix [34].

To sum up, our findings reveal that, even when controlling for production and test code variables, different experience-related metrics turn out to be significant when explaining assertion density. This seems to further demonstrate the role of developers' experience in testing activities and thus also warrants further analyses into this research area.

Summary for RQ$_1$: At least two of the considered experience-related metrics turn out to be statistically significant factors to explain assertion density over the entire dataset. Also, we have found that metrics like WMC and NOM do not frequently correlated to the number of assertions added by developers.

B. *RQ$_2$ - Survey analysis result*

Figure 1 shows the background of our participants. This data comes from the first four questions in Table II. Among the 57 respondents, 39% (22 participants) report that they have more than 5 years of experience in testing activities, while another 42% (24) report between 1 and 3 years of experience. Moreover, 72% of the participants (mostly) work in industry, and 51% (29) of them work in large companies having more than 250 employees. From these descriptive statistics, we can claim that our sample is composed of a variety of developers having enough experience in testing activities and whose opinions are likely to provide us with valid and reliable insights into how developers' experience is correlated to the assertion density. In addition, 32% of the participants work in a large team composed of 5-10 people (18), 21% within a team of 10-12, while the majority (35%) in a small team (*i.e.,* 2-5 people).

Regarding the relevant factors for writing assertions, looking at the boxplots in Figure 2 we see that the majority of them are considered important, as indicated by the median value that is close to 4. For factors such as *Testing Skill*, *Experience within the domain of the project* (*e.g.,* if in the past a developer has worked on similar code and knows where to check for bugs), *Characteristics of Test Code*, and *Presence of Test Smells*, we notice that participants assigned a similar value of importance

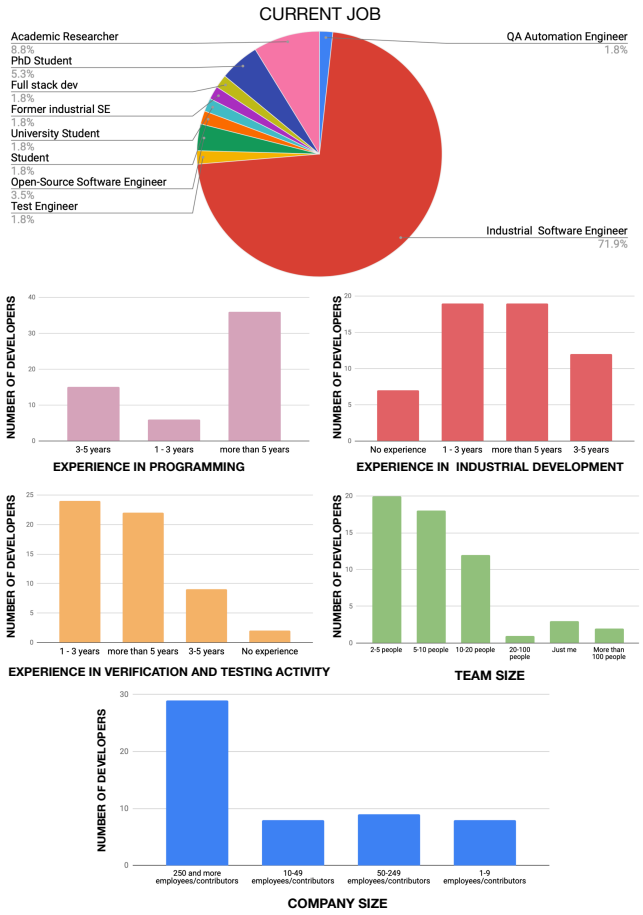

Figure 1: Graphics of the background of our participants

to all of them; this is also visible by looking at the tight shape of the boxplots. Conversely, in the case of *Experience within the Project* (*i.e.,* the knowledge accumulated on the project) and *Characteristics of Production Code*, the shape of the boxplots is extended upward, meaning that participants' answers are mixed but also that some of them consider these factors highly important (many times assigning a value of 5, *i.e.,* extremely important). Finally, we observe that *Developers' authority* is considered as having limited importance to write assertions, *i.e.,* the median value is very low with respect to the others. Note that as a description of this factor, we have provided the following statement: *"Whether the production code has been developed by a core developer or a developer whose reputation is high"*; in doing so, we try to verify if the status of a developer can influence the decisions made during testing activities. The results, however, highlight that our participants are not concerned with this aspect, but rather they base their decisions on good skillset and past experience, as also shown in previous work [70], [71].

From the analysis of the answers provided to the second section of the survey, we can claim that developers consider *experience and code quality as key factors influencing the decision to add test assertions*. Thus, not only the survey confirms our statistical analyses, but also highlights an additional aspect, *i.e.,* the relationship between test code quality [53], [72]–[74]

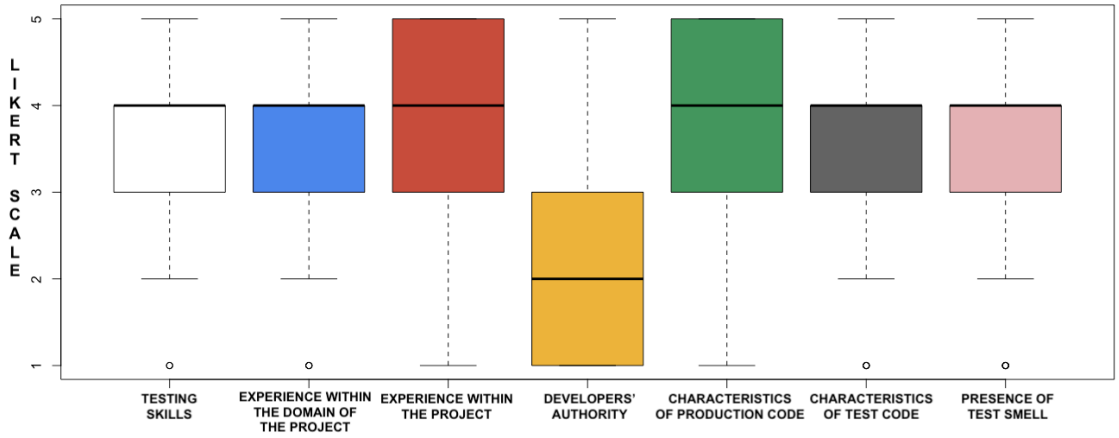

Figure 2: Results related to the relevant aspects for writing assertions.

and assertion density should be further investigated.

Going deeper into the comments left by participants in the last section of the survey, we can provide a more comprehensive view of the usefulness of assertions and the way developers decide to add them in test code. For instance, participant #20 (*i.e.,* an industrial developer that works in a large team of 10-20 people with high experience in verification and testing activity) stated that:

> #20 - *"Assertions make test-driven development possible, which allows immediate feedback to how the application should even run in the first place. They keep bugs from regressing, and when they do pop up, we know how to fix them".*

As such, we can deduce that having a high assertion density is fundamental to properly verifying source code reliability. Still, the mechanism of adding assertions is driven by the experience of the development team. Indeed, as pointed out by one the surveyed developers:

> #15 - *"The experience of who develops tests is necessarily important to make them effective and this influences the way assertions are inserted.".*

Thus, the results of the study seem to corroborate the findings of **RQ**$_1$. Nevertheless, according to our participants, there are also drawbacks when it comes to an excessive usage of the assertion mechanism, as it might create side-effects. More specifically, two of the participants stated:

> #7 - *"Too many assertions in one test makes the test less readable."*

> #12 - *"Testing the right things is hard. More assertions do not automatically mean better tests."*

These opinions somehow recall the concept of *assertion roulette* [72], a test smell which occurs when a test method has multiple non-documented assertions. Multiple assertion statements in a test method without a descriptive message impacts readability/understandability/maintainability as it is harder to understand the reason leading to the failure of the test. Consequently, developers are concerned with the number of assertions to add and try to determine when it is really the case to add a new assert. Moreover, the last statement of participant #12 suggests that assertion density is important but is not the only aspect to consider for test code effectiveness; thus, the quality of tests should be assessed by taking into account additional and complementary aspects such as, for instance, branch or mutation coverage [11]–[13].

In summary, the survey study highlights that, on the one hand, experience represents an important factor when writing assertions, thus corroborating our previous results coming from the statistical modeling done in Section IV-A; on the other hand, our findings also show that assertions need to be treated carefully: although they are widely perceived as a key and useful element for writing effective test cases, over-using them may be detrimental to test design quality.

Summary for RQ$_2$: The survey participants perceive the role of experience as a relevant factor when writing assertions, thus confirming the findings achieved when statistically assessing the impact of experience-related metrics on assertion density. Moreover, they also highlight that test code quality plays a role.

V. Threats to Validity

A number of threats might have influenced our findings. In this section, we summarize and explain how we mitigate them.

A. Threats to construct validity

Threats to *construct validity* are related to the relationship between theory and observation. In our case, they are mainly due to the independent variables used within the statistical model, as well as the dataset exploited in **RQ**$_1$. To compute the team's development and testing experience, we relied on the metrics originally proposed by Kamei *et al.* [37]. We are aware of the existence of other metrics that can capture experience under different angles (*e.g.,* commit-tenure [36], [41]), and we plan to explore their role as part of our future research agenda as well as to measure the phenomenon at a finer granularity (*e.g.,* at individual-developer level). As for the control variables,

we computed them using PyDriller,[8] a publicly available framework able to mine software repositories and compute a number of code metrics. It is important to note that the tool has also been evaluated by the original authors, showing excellent performance [75]. Nevertheless, we cannot exclude possible imprecision in the computation of such variables.

To extract development teams composition, we relied on the definition previously used in literature [35], [36], *i.e.,* a team is the set of developers who have added/modified/removed lines of code to a certain test class. While this definition has been shown to be accurate enough [35], [36], [41], it may be possible that in some cases it is inaccurate because of the presence of developers that are not active anymore in the considered projects, *e.g.,* developers who worked on a test in the past but that are now not part of the team anymore. We recognize this threat to validity: the context selection process, which explicitly selects systems which are still actively developed, partially mitigated the possibility to consider non-active developers.

In the context of $\mathbf{RQ}_2$, threats are related to the way we have measured how important the developer's experience is when writing assertions. In this regard, we have relied on a 5-point Likert scale [65] ranging from *'Not at all important'* to *'Extremely important'*. However, being aware that this indication only gives part of the story, we invited participants to further explain the reported answers, in order to obtain a deeper insight.

B. Threats to conclusion validity

Threats to *conclusion validity* concern the relation between treatment and outcome. A major threat in our context is related to the statistical methods employed. To ensure that the model is appropriate for the available data, we have first investigated how similar studies performed their analyses [76] and verified the assumptions made by the GLM technique on the underlying data. Afterwards, to ensure that the experimented model did not suffer from multi-collinearity, we have adopted the well-established variance inflation factors function [61] to discard non-relevant variables from the considered features, setting the threshold to 5 as suggested by O'Brien [61]. In addition, we have discarded outliers to avoid some interpretation bias [77]. Moreover, we have computed the *R-square* coefficient for evaluating the goodness of fit of our model. Finally, we statistically verified our conjecture while applying some precautions to avoid conclusion biases: in particular, we have defined a number of control variables related to both production and test code [48]. To compute a team-level measure of experience, we have computed the median experience of the developers composing a team: it is worth noting that the results we have achieved were similar even when considering the mean as operator, and thus we can exclude that this choice might have biased our findings.

C. Threats to external validity

Threats to *external* validity relate to the generalizability of the results. In our context, we analyzed 12 systems coming from various ecosystems (*e.g.,* Apache vs Eclipse), having different application domains, and characteristics (size, number of classes, etc.). However, studies aimed at corroborating our findings on a different set of systems would be worthwhile. At the same time, we gathered opinions from 57 developers. Most of them had more than five years of programming and testing experience and, therefore, this limited possible threats to the validity and generalizability of the reported answers. Nevertheless, also in this case a replication with more developers would be beneficial to corroborate our findings.

[8]https://github.com/ishepard/pydriller

VI. Conclusion

The role of developer's experience for software maintenance and evolution tasks has been widely explored in the past [2], [3], [8]–[10]. Similarly, initial compelling evidence showed that experience is also relevant in the context of software testing [1], [4]–[7]. In this paper, we aimed at expanding on this line of research by proposing an empirical study on the role of development teams' experience when writing assertions.

We first performed a quantitative study aimed at understanding the relation of experience to assertion density from a statistical perspective on a set of 12 software systems. Afterwards, we conducted a survey study that triangulated the results and proposed insights coming from 57 developers.

To sum up, this paper proposed the following contributions:

1) Empirical evidence of the relation between the experience of development teams and assertion density of test classes. Our findings suggest that the higher the experience, the higher the number of assertions present in a test. Given previous findings that successfully related assertion density to source code quality [15], our results highlight that the composition of testing teams might represent an important aspect to take into account to assist and improve both software quality and reliability;
2) Insights from practitioners of how more expert developers can produce better test cases, possibly discovering a higher number of faults in production code. Furthermore, our investigation provided hints on the decision-making process followed by developers to decide on whether to add new assertions in test code: this information may be further explored to devise novel recommendation systems that assist developers when writing test code;
3) A replication package containing all data and scripts used to conduct our study [34].

Our findings represent the main input for our future research on the topic. We aim at corroborating our findings on a larger set of systems, possibly considering how the individual developer's experience plays a role in this context. Moreover, it would be also interesting analyze how others factors, *e.g.,* occurrences of test smells or mutation coverage, are correlated to the assertion density. Finally, more analyses on the way developer-related factors influence software testing practices are part of our research agenda, as well as the definition of novel methodologies to help developers writing effective test code.

REFERENCES

[1] R. Pham, S. Kiesling, O. Liskin, L. Singer, and K. Schneider, "Enablers, inhibitors, and perceptions of testing in novice software teams," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 30–40.

[2] G. Catolino, F. Palomba, A. De Lucia, F. Ferrucci, and A. Zaidman, "Enhancing change prediction models using developer-related factors," *Journal of Systems and Software*, vol. 143, no. 9, pp. 14–28, 2018.

[3] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia, "A developer centered bug prediction model," *IEEE Transactions on Software Engineering*, vol. 44, no. 1, pp. 5–24, 2018.

[4] A. Beer and R. Ramler, "The role of experience in software testing practice," in *Euromicro Conf. Softw. Eng. and Advanced Applications (SEAA)*. IEEE, 2008, pp. 258–265.

[5] T. Kanij, R. Merkel, and J. Grundy, "An empirical study of the effects of personality on software testing," in *Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 2013.

[6] ——, "A preliminary survey of factors affecting software testers," in *Australian Softw. Eng. Conf. (ASWEC)*. IEEE, 2014.

[7] J. Rooksby, M. Rouncefield, and I. Sommerville, "Testing in the wild: The social and organisational dimensions of real world practice," *Computer Supported Cooperative Work (CSCW)*, vol. 18, no. 5-6, 2009.

[8] M. Jørgensen and D. I. Sjøberg, "Impact of experience on maintenance skills," *Journal of Software: Evolution and Process*, vol. 14, no. 2, 2002.

[9] J. Li, T. Stålhane, J. M. Kristiansen, and R. Conradi, "Cost drivers of software corrective maintenance: An empirical study in two companies," in *Proceedings of the International Conference on Software Maintenance (ICSM)*. IEEE, 2010, pp. 1–8.

[10] P. Bhatt, G. Shroff, K. Williams, and A. K. Misra, "An empirical study of factors and their relationships in outsourced software maintenance," in *Asia Pacific Software Engineering Conf (APSEC)*. IEEE, 2006, pp. 301–308.

[11] T. T. Chekam, M. Papadakis, Y. Le Traon, and M. Harman, "An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 597–608.

[12] G. Grano, F. Palomba, and H. C. Gall, "Lightweight assessment of test-case effectiveness using source-code-quality indicators," *IEEE Transactions on Software Engineering*, 2019.

[13] Y. Wei, B. Meyer, and M. Oriol, "Is branch coverage a good measure of testing effectiveness?" in *Empirical Software Engineering and Verification*. Springer, 2010, pp. 194–212.

[14] A. Turing, "Checking a large routine," in *The early British computer conferences*. MIT Press, 1989, pp. 70–72.

[15] G. Kudrjavets, N. Nagappan, and T. Ball, "Assessing the relationship between software assertions and faults: An empirical investigation," in *Int'l Symp. on Software Reliability Engineering (ISSRE)*. IEEE, 2006, pp. 204–212.

[16] C. A. R. Hoare, "Assertions: A personal perspective," *IEEE Annals of the History of Computing*, vol. 25, no. 2, pp. 14–25, 2003.

[17] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Future of Software Engineering*. IEEE, 2007, pp. 85–103.

[18] R. B. Johnson and A. J. Onwuegbuzie, "Mixed methods research: A research paradigm whose time has come," *Educational researcher*, vol. 33, no. 7, pp. 14–26, 2004.

[19] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion detection in code reviews," in *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 2017, pp. 549–553.

[20] F. Palomba, D. A. Tamburri, F. A. Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE Transactions on Software Engineering*, to appear.

[21] D. A. Tamburri, F. Palomba, A. Serebrenik, and A. Zaidman, "Discovering community patterns in open-source: a systematic approach and its evaluation," *Empirical Software Engineering*, pp. 1–49, to appear.

[22] D. A. A. Tamburri, F. Palomba, and R. Kazman, "Exploring community smells in open-source: An automated approach," *IEEE Transactions on Software Engineering*, 2019.

[23] D. S. Rosenblum, "A practical approach to programming with assertions," *IEEE Trans. on Softw. Engineering*, vol. 21, no. 1, pp. 19–31, 1995.

[24] B. Meyer, "Applying'design by contract'," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[25] S. Counsell, T. Hall, T. Shippey, D. Bowes, A. Tahir, and S. MacDonell, "Assert use and defectiveness in industrial code," in *Int'l Symp. on Software Reliability Eng. Workshops (ISSREW)*. IEEE, 2017, pp. 20–23.

[26] S. McConnell, *Code complete*. Pearson Education, 2004.

[27] L. A. Clarke and D. S. Rosenblum, "A historical perspective on runtime assertion checking in software development," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 3, pp. 25–37, 2006.

[28] J. Chen, Y. Bai, D. Hao, L. Zhang, L. Zhang, and B. Xie, "How do assertions impact coverage-based test-suite reduction?" in *Int'l Conf. on Software Testing, Verification and Validation (ICST)*. IEEE, 2017.

[29] P. S. Kochhar and D. Lo, "Revisiting assert use in GitHub projects," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, 2017, pp. 298–307.

[30] C. Casalnuovo, P. Devanbu, V. Filkov, and B. Ray, "Replication of assert use in GitHub projects," *Methods*, vol. 947, no. 770,175, pp. 1–717, 2015.

[31] C. Casalnuovo, P. Devanbu, A. Oliveira, V. Filkov, and B. Ray, "Assert use in GitHub projects," in *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 2015, pp. 755–766.

[32] G. Catolino, F. Palomba, F. A. Fontana, A. De Lucia, A. Zaidman, and F. Ferrucci, "Improving change prediction models with code smell-related information," *arXiv preprint arXiv:1905.10889*, 2019.

[33] C. Vassallo, G. Grano, F. Palomba, H. C. Gall, and A. Bacchelli, "A large-scale empirical exploration on refactoring activities in open source software projects," *Science of Computer Programming*, 2019.

[34] Anonymous. (2019) How the experience of development teams influences the assertion density of test classes - online appendix https://figshare.com/s/c3d4729efc978debfeb4.

[35] A. Michailidou and A. Economides, "Gender and diversity in collaborative virtual teams," *Computer Supported Collaborative Learning: Best practices and principles for instructors*, pp. 199–224, 2008.

[36] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in GitHub teams," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, 2015, pp. 3789–3798.

[37] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, 2013.

[38] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. ACM, 2014, pp. 192–201.

[39] G. Catolino, "Just-in-time bug prediction in mobile applications: the domain matters!" in *Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2017, pp. 201–202.

[40] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. ACM, 2014, pp. 172–181.

[41] G. Catolino, F. Palomba, D. A. Tamburri, A. Serebrenik, and F. Ferrucci, "Gender diversity and women in software teams: How do they affect community smells?" in *41st International Conference on Software Engineering, Software Engineering in Society*, 2019.

[42] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[43] S. Jungmayr, "Testability measurement and software dependencies," in *Proceedings of the 12th International Workshop on Software Measurement*, vol. 25, no. 9, 2002, pp. 179–202.

[44] L. Badri, M. Badri, and F. Toure, "An empirical analysis of lack of cohesion metrics for predicting testability of classes," *International Journal of Software Engineering and Its Applications*, vol. 5, no. 2, pp. 69–85, 2011.

[45] N. I. Churcher, M. J. Shepperd, S. Chidamber, and C. Kemerer, "Comments on" a metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 21, no. 3, 1995.

[46] E. J. Weyuker, "Evaluating software complexity measures," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1357–1365, 1988.

[47] D. Athanasiou, A. Nugroho, J. Visser, and A. Zaidman, "Test code quality and its relation to issue handling performance," *IEEE Transactions on Software Engineering*, vol. 40, no. 11, pp. 1100–1125, 2014.

[48] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, 1994.

[49] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[50] A. H. Watson, D. R. Wallace, and T. J. McCabe, *Structured testing: A testing methodology using the cyclomatic complexity metric*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1996, vol. 500, no. 235.

[51] B. Henderson-Sellers, L. L. Constantine, and I. M. Graham, "Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design)," *Object oriented systems*, vol. 3, no. 3, pp. 143–158, 1996.

[52] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto, and A. De Lucia, "Automatic test case generation: What if test code quality matters?" in *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 2016, pp. 130–141.

[53] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli, "On the relation of test smells to software code quality," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 1–12.

[54] E. Fregnan, T. Baum, F. Palomba, and A. Bacchelli, "A survey on software coupling relations and tools," *Information and Software Technology*, 2018.

[55] M. Egger, G. D. Smith, M. Schneider, and C. Minder, "Bias in meta-analysis detected by a simple, graphical test," *Bmj*, vol. 315, no. 7109, pp. 629–634, 1997.

[56] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 435–445.

[57] Y. K. Malaiya, M. N. Li, J. M. Bieman, and R. Karcich, "Software reliability growth with test coverage," *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 420–426, 2002.

[58] J. A. Nelder and R. J. Baker, *Generalized linear models*. Wiley Online Library, 1972.

[59] U. Halekoh, S. Højsgaard, J. Yan *et al.*, "The r package geepack for generalized estimating equations," *Journal of Statistical Software*, vol. 15, no. 2, pp. 1–11, 2006.

[60] G. D. Garson, "Testing statistical assumptions," *Asheboro, NC: Statistical Associates Publishing*, 2012.

[61] R. M. O'brien, "A caution regarding rules of thumb for variance inflation factors," *Quality & quantity*, vol. 41, no. 5, pp. 673–690, 2007.

[62] D. Zhang, "A coefficient of determination for generalized linear models," *The American Statistician*, vol. 71, no. 4, pp. 310–316, 2017.

[72] A. Van Deursen, L. Moonen, A. Van Den Bergh, and G. Kok, "Refactoring test code," in *Proceedings of the 2nd international conference on extreme*

[63] D. E. Ramo, S. M. Hall, and J. J. Prochaska, "Reliability and validity of self-reported smoking in an anonymous online survey with young adults." *Health Psychology*, vol. 30, no. 6, p. 693, 2011.

[64] C. E. Nielsen, "Coproduction or cohabitation: Are anonymous online comments on newspaper websites shaping news content?" *New Media & Society*, vol. 16, no. 3, pp. 470–487, 2014.

[65] P. M. Chisnall, "Questionnaire design, interviewing and attitude measurement," *Journal of the Market Research Society*, vol. 35, no. 4, pp. 392–393, 1993.

[66] T. S. Flanigan, E. McFarlane, and S. Cook, "Conducting survey research among physicians and other medical professionals: a review of current literature," in *Proceedings of the Survey Research Methods Section, American Statistical Association*, vol. 1, 2008, pp. 4136–47.

[67] H. Abdeen, S. Ducasse, and H. Sahraoui, "Modularization metrics: Assessing package organization in legacy large object-oriented software," in *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE, 2011, pp. 394–398.

[68] W. W. Chin *et al.*, "The partial least squares approach to structural equation modeling," *Modern methods for business research*, vol. 295, no. 2, pp. 295–336, 1998.

[69] J. Cohen, *Statistical power analysis for the behavioral sciences*. Routledge, 2013.

[70] S. Schachter, "The psychology of affiliation: Experimental studies of the sources of gregariousness." 1959.

[71] R. P. Bagozzi and U. M. Dholakia, "Open source software user communities: A study of participation in linux user groups," *Management science*, vol. 52, no. 7, pp. 1099–1115, 2006.
programming and flexible processes in software engineering (XP2001), 2001, pp. 92–95.

[73] G. Meszaros, *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[74] F. Palomba and A. Zaidman, "The smell of fear: On the relation between test smells and flaky tests," *Journal of Empirical Software Engineering*, 2019.

[75] D. Spadini, M. Aniche, and A. Bacchelli, "Pydriller: Python framework for mining software repositories," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 908–911.

[76] B. Winter, "A very basic tutorial for performing linear mixed effects analyses," *arXiv preprint arXiv:1308.5499*, 2013.

[77] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.