Article

An Expert Judgment in Source Code Quality Research Domain—A Comparative Study between Professionals and Students

Luka Pavlič * , Marjan Heričko and Tina Beranič

Faculty of Electrical Engineering and Computer Science, University of Maribor,
Koroška Cesta 46, 2000 Maribor, Slovenia; marjan.hericko@um.si (M.H.); tina.beranic@um.si (T.B.)

* Correspondence: luka.pavlic@um.si; Tel.: +386-2-220-7356

Received: 6 September 2020; Accepted: 2 October 2020; Published: 12 October 2020

Abstract: In scientific research, evidence is often based on empirical data. Scholars tend to rely on students as participants in experiments in order to validate their thesis. They are an obvious choice when it comes to scientific research: They are usually willing to participate and are often themselves pursuing an education in the experiment's domain. The software engineering domain is no exception. However, readers, authors, and reviewers do sometimes question the validity of experimental data that is gathered in controlled experiments from students. This is why we will address this difficult-to-answer question: Are students a proper substitute for experienced professional engineers while performing experiments in a typical software engineering experiment. As we demonstrate in this paper, it is not a “yes or no” answer. In some aspects, students were not outperformed by professionals, but in others, students would not only give different answers compared to professionals, but their answers would also diverge. In this paper we will show and analyze the results of a controlled experiment in the source code quality domain in terms of comparing student and professional responses. We will show that authors have to be careful when employing students in experiments, especially when complex and advanced domains are addressed. However, they may be a proper substitution in cases, where non-advanced aspects are required.

Keywords: comparative study; expert judgment; performing experiments; source code quality; student participants

1. Introduction

Software quality is a well-defined and accepted term within the software engineering domain. Within software development teams there needs to be a strong consensus about not only what software quality is, but also on how to measure it. Several formal [1] and de-facto standards list different software quality attributes and prescribe procedures and metrics for software quality evaluation. They range from relatively simple indirect methods for measuring source code in order to be confident in the expected quality (e.g., comment-to-code ratio) to relatively complex processes. Requiring several ceremonies, many people, is time-consuming and is described in standards, like in ISO/IEC 25040 [2].

While software quality is a strong research domain, the evaluation of new or improved approaches according to existing ones is essential in order to ensure a high level of quality within the software engineering domain [3]. This can be done by using different empirical research approaches, including the implementation of experiments, whose importance and widespread use within software engineering has already been highlighted in many studies [3–5].

Moreover, source code quality, in addition to being measured and exposed to certain threshold values, is usually assessed by ad-hoc or systematic code reviews. With the industry-wide introduction

of distributed configuration management systems, such as Git [6], and the development process, code reviews have become the norm in the industry. An example of this is the obligatory code review during/before pull requests. In this context, subjective judgment on source code quality (e.g., size, complexity, cohesion) has priority over automated code measurements, which, ideally would be performed before a code review is triggered. Code reviews, i.e., professional source code evaluation, boast several benefits. They include an evaluation of the source code by more experienced developers (usually “seniors” review the source code of “juniors”), which has so far been an utopia in the software quality assurance domain.

Since source code quality is a well-defined and understood term, one would expect that source code evaluation would be straightforward and a formal education with some practical experience would be more than enough to operate with it. In this paper we will clearly show that in addition to formal education (students), experience (professionals) is crucial.

Several researchers have been advocating their findings and novel approaches in the software engineering area by not experimenting with industry-level professional developers. Instead, groups of students are employed and usually do not question experimental results, e.g., when we want to judge whether the class structure has appropriate inheritance depth, asking final-year IT students that passed Object Orientation and Software Quality Assurance courses. An evaluation of the approach, would be more than appropriate.

However, the approach could be questioned with some other, more advanced evaluations. In this paper we would like to validate, systematically, if and when student groups can replace groups of professionals in order to support research outcomes in the area of software quality. We believe that in order to provide reliable results, one of the essential parts of an experiment is the participating subjects. The subjects need to have not only in-depth knowledge of the domain, wherein this has to be valid for a piece of the overall knowledge but also for the specific knowledge covering the experiment’s domain. The use of professionals versus students is a known dilemma [7,8]. Since the professionals are usually hard to motivate to participate, students are, especially in the academic domain, more readily available participants. However, their knowledge, not to mention day-to-day experience, is usually not specific enough, which may result in low external validity [7].

This paper presents a study that investigates the impact of participants’ experiences involved in the expert judgment in software quality and software metrics research. Our goal was to answer the following research questions:

1. Are participants consistent in expert judgment evaluations regarding the source code quality research domain?
 - (a) What is the level of agreement between students in their expert judgments regarding the source code quality research domain?
 - (b) What is the level of agreement between professionals in their expert judgments regarding the source code quality research domain?
2. Are the levels of agreement between students and the level of agreement between professionals comparable?
 - (a) In which aspects of source code quality assessment are the levels of agreement comparable?

By answering our research questions, we would like to gain a clear insight into the dilemma of employing students in experiments, with regard to source code quality domain, without compromising the external validity of the results. In addition to this, we would also like to provide advice in the form of when experience (professionals) is crucial and when a formal education with minor experience (students) is enough.

The paper is organized as follows. In the next section, we will summarize the research background and outline the most important related work in the domain. In Section 3 we present hypotheses and

the research method in detail. Section 4 gives an overview of our experiment, presents tasks that participants are exposed to, and summarizes the experiment results. In Section 5, we discuss the experiment results, while outlining the differences between students and professionals. We discuss research questions and conclude this paper with Section 6.

2. Research Background and Related Work

An application of empirical research methods has been researched by many papers. Zhang et al. [9] present a mapping study, wherein in the majority of selected studies, the experiment is used as an empirical method. As the results show, experiments are followed by case studies, surveys, literature reviews, replication experiments, pilot studies, and simulations [9]. Furthermore, the expert judgments are not a frequently used approach within software engineering experiments.

The scope of using experiments within software engineering and its sub domains has spread over the years. In a literature review provided by Sjöberg et al. [10], only one study was detected that implemented the experiment in the domain of software metrics and measurement. On the other hand, 13 years later, Zhang et al. [9] identified 48 studies, with 16.2%, implementing an experiment in the software quality domain. The use of experiments in software engineering started in the 1960s [3]. The authors [3] list four dimensions characterizing the context of the experiment, including students vs. professionals, opening up a challenging domain. As the literature review provided by Sjöberg et al. [10] shows, professionals were used as the experimental subjects in only 9% of studies, and on the other hand, 86.8% of studies used students. Undergraduate students are used much more frequently, whereas graduate students are used in 10.8% of studies [10]. Sjöberg et al. [10] also reported that only three studies in seven used students and professionals while measuring the differences between the groups. No research detected differences. This was also confirmed by Daun et al. [4], conducting a systematic mapping study looking into the state of the art in controlled experiments with students in software engineering. As the results indicate that the majority of controlled experiments are done using student participants, 42.33% only used graduated students. In 15.95% of experiments, students received evaluation tasks, wherein the most frequently experiments researched the student's comprehension skills [4].

Falessi et al. [7] presented the positive and negative aspect of using students and professionals in the experiments. The professionals are more challenging to acquire than students, and when they are willing to cooperate, the sample size is usually small [7]. However, the external validity of the results is better in contrast to using students [7]. As the results show, the choice of an appropriate subject depends upon understanding the developer population portion represented by the participants [7]. As added by Feldt et al. [11], student samples could efficiently step-in for a specific subset of professionals. Falessi et al. [7] also propose a characterization scheme dividing subjects based on their experience. The experiences can be described using three dimensions: real, relevant and recent [7]. The fact that the use of the terms professionals and students may be misleading was also pointed by Feitelson [8], who at the same time exposed the problem of using years of experiences as a metric. Feitelson [8] lists some drawbacks when using students as an experiment's subject. As the results of the literature review show, students could not represent the professionals, namely due to a lack of experience, differences in the use of technology, learning misconceptions, and academic orientation that may not be aligned with professional practice [8].

Within the experiments, the use of expert judgment for evaluation could also be used. Expert judgment is frequently used as one of the estimation techniques within the project management [12,13], wherein experts have specialized knowledge [14]. Expert judgment within software quality presents the use of developers' experiences for reliable evaluation [15]. Boehm [12] defines expert judgment as the consultation of one or more experts. Hughes [13] added that experts possess experiences familiar to the judging domain. Since the uniqueness of software products makes quality evaluation a challenging task [15], the domain of experts' experiences is crucial.

Expert judgment is also used in the code smells domain, for example, when investigating the code smells impact on system level maintainability [16]. Bigonha et al. [17] used manual inspection in order to validate the code smells provided by the tool. Oliveira et al. [18] validated derived thresholds with the help of developers. Moreover, Rosqvist et al. [19] presented a method for software quality evaluation using expert judgment. Within expert judgements, evaluators form their opinion based on past experiences and knowledge, which can result in the subjectivity of the provided assessments [13,19]. Rosqvist et al. [19] claim that each expert judgement is based on a participant's mental model that is used to interpret the assessed quality aspect. However, considering and properly addressing the mentioned challenges, experts' assessments can—despite being based on participants' personal experience—constitute a good and valuable supplement to empirical shreds of evidence [19].

While many studies address the use of students in software engineering experiments in general, the characteristics of the subjects involved in expert judgment were not addressed yet. As Falessi et al. [7] says, the comparison of the performance of professionals and students is enabled when experiments are implemented using both types of participants. This was done in our study, where the expert judgment in the source code quality research domain was done using students and experts. Additionally, we did not find research that would address the question of substituting professional participants with students in terms of performing empirical research, as is the case with the research that we present in this paper.

3. Research Method

In order to answer the research questions, an experiment was designed. We experimented separately with professionals and students. A bird-eye view of our approach is shown in Figure 1. As illustrated, we wanted to capture individual as well as coordinated evaluations for the same set of source code entities—once by employing professionals, then again with the student group. The details of our single experiment are shown in Figure 2.

Before beginning, all participants provided their profile. We designed our questionnaire based on the practices set forth Chen et al. [20]. We asked them to enter their perceived level of knowledge of programming languages and provide a number of years for their professional experience. Since the knowledge self-assessment can be biased and subjective, the years of experience criterion was added in order to objectify participant's experiences. In addition, to have a record of a participant's classification in terms of students and professionals, it also gives us an opportunity to check whether participant profiles are comparable.

Figure 1. Bird-eye view on the implemented research.

Figure 2. The experiment process—the experiment was executed for students and professionals separated.

The participants were asked to evaluate several aspects of source code quality, i.e., class size, class complexity, class cohesion, coupling with other classes and general quality assessment. The example of an evaluation form for a software class is presented in Table 1. The participant evaluated each aspects using the scale: “very poor”, “poor”, “good”, “very good”. The scale aims at gathering their opinion about the quality of the assessed software entity, e.g., during the source code size assessment, the evaluators assessed, if, in their opinion, the size of a software class is poor (i.e., inappropriate) or good (i.e., appropriate). A software class where the source code size is evaluated as poor, contains too many or too little lines of code, resulting in unmanageable size and opacity or, on the other hand, in inappropriately short content. Contrarily, a software class that is assessed as good in the terms of source code size, is composed of manageable and acceptable lines of code.

Table 1. A program entity evaluation form.

Entity: net.sf.jasperreports.engine.export.JRTextExporter	
Class	
JRTextExporter.java	class source code [21]
Assessment	
general assessment	☐ very poor ☐ poor ☐ good ☐ very good
source code size	☐ very poor ☐ poor ☐ good ☐ very good
class complexity	☐ very poor ☐ poor ☐ good ☐ very good
class cohesion	☐ very poor ☐ poor ☐ good ☐ very good
coupling with other classes	☐ very poor ☐ poor ☐ good ☐ very good
Comment	
assessor comment	

An example of a software class assessment is shown in Tables 2 and 3. Table 2 depicts the assessment and coordination of the chosen software class. After the individual evaluations of each participant were made (shown in Table 1 as Assessor 1 and Assessor 2), the participants were asked to coordinate their evaluations with the assigned co-assessor, providing a coordinated and agreed-upon final evaluation (shown in Table 1 as Coordinated). For example, Assessor 1 assessed the source code size as “very good” and Assessor 2 assessed the same quality aspect as “poor”, they have to coordinate their chosen assessments. Based on the exchanged views they assessed the source code size of evaluated software class as poor. We forced participants to coordinate their assessments as a measure

to get as objective assessments as possible on one hand, and to address possible inconsistencies in the assessor's subjective views.

Table 2. An example of program entity assessment and coordination.

Entity: net.sf.jasperreports.engine.export.JRTextExporter				
Assessment	Assessor 1	Assessor 2		Coordinated
general assessment	● very good	● very good	⇒	● very good
source code size	● very good	● poor	⇔	● poor
class complexity	● poor	● very good	⇔	● very good
class cohesion	● poor	● very good	⇔	● poor
coupling with other classes	● poor	● poor	⇒	● poor

Table 3. An example of cross section between students and between students and professionals in categories general assessment (1), source code size (2), class complexity (3), class cohesion (4) and coupling with other classes (5).

Entity: net.sf.jasperreports.engine.export.JRTextExporter										
students ∩ students					students ∩ professionals					
(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)	
■	■	■	■	n/a	■	■	■	■	n/a	
■	■	■	■	n/a	■	■	■	■	n/a	
■	■	■	■	■	■	■	■	■	n/a	
■	■	■	■	■	■	■	■	■	n/a	
■	■	■	■	■	■	■	■	■	n/a	

The subset from the same repository of source code entities was given to every participant. This is how each participant had to evaluate several, but not all entities. At the same time, each entity was evaluated by several participants. They were asked to provide their evaluations. We asked participants to evaluate, how appropriate they found the provided source code in terms of size, complexity, cohesion, coupling and how they evaluate overall source code quality. They had to judge each aspect in the 4-step range from “very poor” to “very good”.

After all the participants finished their judgments, groups were formed, based on source code entities. This means that each participant was a member of several groups. One member was elected per group to be a group leader, which, once an agreement on a joint entity evaluation was reached, confirmed the agreed-upon evaluation. Group members had to collaborate and discuss only the source code quality aspects, where they did not provide the same judgment during their evaluation session. Since experiments with students and professionals were executed separately, there was no coordination group with mixed participants (students and professionals). This is why coordinated evaluations, which would separate student and professional participants are also a useful source of data.

In order to conclude all data records, including participant profiles, individual evaluations, and coordinated evaluations (that clearly marked which evaluators changed their decision during the collaborative evaluation step), were combined into a report. Please note that the experiment was performed separately for professionals and students, but the source code entity repository was the same. This allowed us to not only observe inconsistencies within the groups, but also to observe them between groups, and, more importantly, to compare student and professional efforts, results and group agreement.

4. The Case: Expert Judgment in Software Quality Domain

4.1. Source Code Quality Judgment Tool

The expert judgments were performed using the developed source code assessment tool. The tool supported coordination between assessors in order to reduce bias and achieve greater reliability for

evaluations. Figure 3 demonstrates examples of a cross-section between participants. In the first step of the evaluation process, individual assessments are made.

Figure 3. A screenshot from the tool, demonstrating the progress of assessments.

Individual assessments of entities are coordinated between linked assessors within the coordination step.

The architecture of the developed tool is presented in Figure 4. The tailor-made IT solution consists of four parts. The external Single-Sign-On provider provides authentication. Components are containerized—front-end, back-end and persistent storage are placed into a docker, communicating via REST web interfaces. The Evaluators Rich Web Application could be extended with additional functionalities. The back-end system covers the entity and assessment management, assessment implementation and reporting and is connected to a sustainable data repository covering the safe and distributed data storage.

The tool was designed in an extendable and adjustable way. The concept allows the use of the tool in different domains by including additional components and making the needed adjustments. Therefore, instead of source code, the tool could also be used for the evaluation of any other entities, since the assessment criteria and number of assessors could be freely adapted.

Figure 4. Source Code Quality Judgment Tool Architecture.

4.2. The Results

In order to get a more detailed profile of participating experts and students, we gathered their experiences and perceived knowledge. The perceived knowledge of the programming language was self-assessed, since Feitelson [8] presents it as a good option for assessing proficiency. The profile of participants is presented in Table 4. In the study, 54 students and 11 experts participated. Participating students evaluated their experiences with software development with an average score of 4 on a scale from 1 to 10, while experts evaluated their experiences with an average of 8.1. On average, students evaluated their knowledge of Java with 4.9 and experts with 8.6. The difference between students and experts can also be seen in their years of experience with Java. While most of the students had less than three years of experience, the majority of experts had more than ten years of experience.

Table 4. Profile of participating students and experts.

	Students	Experts
Number of participants	54	11
Experiences with software development (1–10)	4.0	8.1
Knowledge of Java (1–10)	4.9	8.6
Years of experiences with Java	between 3 and 6 years (31%) less than 3 years (69%)	more than 10 year (81%) between 6 and 10 years (19%)

In the implemented study, participants evaluated 33 different program entities in the form of Java Classes. Among the experts, 16 entities were evaluated by three pairs of assessors, and 15 entities by two pairs of assessors. One entity was evaluated by four pairs of assessors and one entity by one pair. On the other hand, among students, 10 entities were evaluated by two pairs of assessors, 7 by three pairs, 6 entities were evaluated by four pair of assessors, 5 by five pairs, 3 entities by six pairs and 2 by one pair of assessors. The assessors were randomly divided into pairs, during each assessor's assessment of between 7 and 9 entities.

4.3. Results Analysis

As we are interested in examining and comparing the student's and expert's judgments of the source code quality and software metrics, computing the agreement between participants (in our case, this would be an agreement in the judgment of pairs of students and experts) would be most suitable approach for an analysis. Several measures of inter-participant agreement exist. However, they are mainly limited to estimating the agreement between the two participants. In our case, more than one participant gave their assessment of the source code. Furthermore, they assessed it on a dichotomous scale. To enable the estimation of inter-participant agreement in such cases (more than

two participants), the Fleiss's kappa was used, which was first developed in 1971 [22]. Fleiss's kappa has since proven itself to be a useful tool to estimate inter-participant agreement between several participants in the literature [23].

To answer our research questions, we followed the example of Denham [23], and first calculated the agreement proportions for each entity separately using the following equation:

$$P_i = \frac{1}{n(n-1)} \sum_{j=1}^k n_{ij}(n_{ij} - 1),$$

where i represents the number of subjects (in case of calculations for a specific entity, this would be 1), n represents the number of participants, and n_{ij} represents the number of participants assigned to the i th subject in the j th category [22,23]. Specifically, Fleiss's kappa was calculated for each entity on the expert's and student's general assessments of the source code, the assessment of source code size, class complexity, class cohesion, and coupling with other classes. Furthermore, the generalized Fleiss's kappa for each assessment category was calculated, representing the mean category assessment across all entities. The equation, provided by Denham [23] was applied,

$$\frac{1}{N} \sum_{i=1}^N P_i$$

where N represents the total sum of subjects (e.g., entities) and P_i represents the kappa value for the i th entity. The results can be interpreted in terms of the level of agreement between participants. Recommendations on the interpretation of the kappa that are given by Landis and Koch [24], can be found in Table 5.

Table 5. Interpretation of the Fleiss's kappa.

k	Interpretation
>0	Poor agreement
0.01–0.20	Slight agreement
0.21–0.40	Fair agreement
0.41–0.60	Moderate agreement
0.61–0.80	Substantial agreement
0.81–1.00	Almost perfect agreement

We calculated Fleiss's kappa values for each program entity for both students and professionals. The data, interpreted as shown in Table 5, is collected in Table 6. Please note that the table shows the agreements of professional or student groups on a specific aspect of the given entity. The "n/a" value in the table means, that the group did not assess the given aspect of the entity. The "-" value in the table means that there was only one evaluation of the given aspect, which is why the calculation of Fleiss's kappa value was not performed.

Table 6. Values of Fleiss's kappa for each program entity displayed for (1) overall quality, (2) size, (3) complexity, (4) cohesion and (5) coupling.

Entity	Experts					Students				
	(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)
E1	1.00	1.00	1.00	n/a	1.00	0.50	1.00	1.00	0.33	0.50
E2	1.00	1.00	1.00	1.00	n/a	1.00	1.00	1.00	0.50	1.00
E3	1.00	1.00	1.00	1.00	n/a	0.33	0.33	0.33	0.33	0.33
E4	1.00	1.00	1.00	1.00	n/a	-	-	-	-	-
E5	1.00	1.00	1.00	1.00	n/a	1.00	1.00	1.00	1.00	n/a
E6	1.00	1.00	1.00	n/a	1.00	1.00	1.00	0.60	n/a	0.40

Table 6. Cont.

Entity	Experts					Students				
	(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)
E7	1.00	1.00	1.00	1.00	n/a	1.00	1.00	1.00	0.40	1.00
E8	1.00	1.00	1.00	n/a	1.00	0.47	0.67	0.47	0.50	0.47
E9	1.00	1.00	1.00	n/a	1.00	1.00	1.00	1.00	n/a	1.00
E10	1.00	1.00	1.00	n/a	1.00	1.00	1.00	1.00	n/a	1.00
E11	1.00	1.00	1.00	1.00	n/a	0.33	1.00	0.33	1.00	1.00
E12	1.00	1.00	1.00	n/a	1.00	1.00	1.00	1.00	n/a	1.00
E13	1.00	1.00	n/a	1.00	1.00	0.60	1.00	1.00	0.40	0.40
E14	1.00	1.00	1.00	1.00	n/a	1.00	1.00	1.00	1.00	n/a
E15	1.00	1.00	1.00	1.00	n/a	1.00	1.00	1.00	0.50	0.33
E16	1.00	1.00	1.00	1.00	n/a	1.00	0.67	1.00	0.67	0.60
E17	1.00	1.00	1.00	1.00	n/a	0.40	1.00	0.40	0.60	0.33
E18	1.00	1.00	1.00	n/a	1.00	1.00	1.00	1.00	n/a	1.00
E19	1.00	1.00	1.00	1.00	n/a	-	-	-	-	-
E20	1.00	1.00	1.00	n/a	1.00	1.00	1.00	1.00	n/a	1.00
E21	1.00	1.00	1.00	n/a	1.00	0.33	0.33	0.33	0.33	0.33
E22	1.00	1.00	1.00	1.00	n/a	0.33	0.33	1.00	1.00	1.00
E23	1.00	1.00	n/a	1.00	1.00	0.33	0.33	0.33	1.00	0.33
E24	1.00	1.00	n/a	1.00	1.00	1.00	n/a	0.33	n/a	0.33
E25	-	-	-	-	-	1.00	0.33	0.33	1.00	1.00
E26	1.00	1.00	1.00	n/a	1.00	0.67	0.67	0.67	0.60	0.67
E27	1.00	1.00	1.00	n/a	1.00	1.00	1.00	0.60	0.50	0.60
E28	1.00	1.00	1.00	1.00	n/a	0.50	0.50	0.50	0.33	0.33
E29	1.00	1.00	1.00	n/a	1.00	1.00	1.00	1.00	1.00	0.33
E30	1.00	1.00	1.00	n/a	1.00	0.50	1.00	1.00	0.00	0.33
E31	1.00	1.00	1.00	1.00	n/a	0.00	1.00	0.00	0.00	1.00
E32	1.00	1.00	1.00	n/a	1.00	0.00	1.00	0.00	0.00	1.00
E33	1.00	1.00	n/a	1.00	1.00	1.00	1.00	1.00	0.00	1.00
Mean	1.00	1.00	1.00	1.00	1.00	0.72	0.84	0.72	0.54	0.68

5. Discussion

Table 6 summarizes the data for the interpretation and the discussion. Having presented result data, it seems quite obvious, that, unlike in a professional-driven experiment, a student-driven experiment with the same setting, resulted in data, that diverges both when we compare them in a student-to-student manner as well as when we compare the data in a student-to-professional manner (see Table 6).

Based on the established Fleiss's kappa value interpretation [22], it is obvious not only that strong agreement is achieved inside a professional group (highest value, i.e., “almost perfect agreement” is reached practically on all evaluated aspects), but also, that the agreement is consistent throughout all quality aspects that we included in the experiment (“almost perfect agreement” according to [22]).

On the other hand, the Fleiss's kappa values for student participants show worse performance in terms of the achieved agreement in practically all aspects. We can also find entities (E30–E33) with one or several aspects where poor agreement (the lowest rate according to [22]) was recorded.

To support our findings further, the aggregated values of Fleiss's kappa for all entities was also calculated, as shown in Table 7.

Based on the aggregated Fleiss's kappa values in Table 7, it is even more evident that there is a lack of agreement inside the student group (the student group diverges in their answers).

As Fleiss's kappa values and their established interpretation clearly shows, professionals are consistent in their expert judgments. Furthermore, as shown in Table 7, professionals can reach almost perfect agreement (the highest value 1.0, see Tables 5 and 7) on all given source code quality aspects. In the “cohesion” aspect, professionals reached the worst agreement (value 0.8) which is just below “Almost perfect agreement” and is the highest value in the interpretation of “substantial agreement”.

However, as we will show, the “cohesion” aspect was shown to be the most complex, since students also had the lowest agreement on it.

Table 7. Values of Fleiss’s kappa for the measured quality aspects.

Experts					
	Overall quality	Size	Complexity	Cohesion	Coupling
κ	1.00	1.00	1.00	0.80	1.00
Students					
	Overall quality	Size	Complexity	Cohesion	Coupling
κ	0.61	0.81	0.65	0.50	0.53

The situation, when comparing agreement levels, is different when looking at student groups. As shown in Table 7, almost perfect agreement is found only when we take into account source code size. Furthermore, even in this case, the value is the lowest possible one, to be interpreted as “almost perfect agreement” (0.81). In other aspects, agreement barely achieves a substantial level (complexity with a value of 0.65, overall quality with the lowest value for this agreement class—0.61). It is only the “moderate agreement” level where we can observe cohesion (0.50) and coupling (0.53). This is why we cannot state that students are consistent in their expert judgments with the source code quality research domain. However, we can identify a particular aspect where student answers do not diverge.

Based on this evidence, we can answer research question 1: “Are participants consistent in expert judgment evaluations regarding the source code quality research domain?”. In the case of professional participants, where strong experience is reported, based on an interpretation of the Fleiss’s kappa values, we can answer research question 2b as positive. Professionals are consistent in their expert judgment regarding the source code quality research domain. Their level of agreement is almost perfect. In the case of student participants, where domain education seems strong, but a lack of experience is reported, the answer is not straightforward. The interpretation of the Fleiss’s kappa values shows lower agreement levels for all aspects (research question 2a). In the “source code size” aspect, the level of agreement inside student groups reached the same level (based on Fleiss’ kappa values interpretation) as in professional groups (i.e., “Almost perfect agreement”). The lowest level of agreement was seen in the “cohesion” and “coupling” aspects.

This brings us to the final discussion on research question 2: “Are the levels of agreement between students and level of agreement between professionals comparable?” Based on the interpretation of Fleiss’s kappa values (see Tables 7) we can report important differences in the majority of aspects. The exception is, when participants judged the “source code size” aspect. This is why we cannot easily answer research question 2—the level is comparable only in certain aspects, for others it is not. This is why we can the distinct quality aspects while answering research question 2a: “In which aspects of source code quality assessment is the level of agreement comparable?”.

When we observe the “size” and “complexity” aspects, students performed the best in terms of converging to the same answer. Our interpretation of this result is, that those aspects are an example of simple questions, where experience does not play an important role (“Do you find this source code to be an appropriate size?”/“Do you find this source code to be of appropriate complexity?”). However, to judge more complex aspects, such as “overall quality”, “coupling” and, even more apparent, “cohesion”, the experience seems to be necessary, thus students are not an appropriate substitution while experimenting.

To sum up: our research is detailed in its finding that students are an appropriate substitution for professional participants in experiments in the source code quality research domain only when the experiment is dealing with simple aspects (e.g., source code size, source code complexity). When dealing with complex aspects, participants should show a certain level of professional experience in order to keep the experiment sound and relevant in terms of external validity.

6. Conclusions

In this paper we reported our research in order to question whether students are a comparable substitute for professionals during experiments in the source code quality research domain. On several occasions, scholars have tended to use students as an important source of participants in experiments to show and validate their thesis. As we showed during the paper, there are numerous reasons for this, one of the most important being that students are usually willing to participate and are pursuing an education in the experiment's domain. Other authors have also expressed their doubts on the external validity of student-based experiments. This is why we addressed that question in this paper. While performing an experiment, gathering results, interpreting them and answering our research questions, we showed where and why students can or cannot replace professionals. Professionals are participants that boast higher level of experience, in contrast to students, whose education level might be high but who are often missing a comparable level of experience.

We designed and performed an experiment in order to answer the research questions. We experimented separately with professionals and students. The tool, which was tailor-made for this experiment, supported coordination between assessors in order to reduce bias and achieve greater reliability for the evaluations. Based on the results analysis we showed that professionals achieved almost perfect agreement for all given source code quality aspects. On the other hand, when we dealt with student groups, almost perfect agreement was found only when we took into account source code size, which is a representative of asking an experiment participant a simple question.

Based on our findings, students were not out-performed by professionals in certain aspects, and students would mostly not only give different answers compared to professionals, but their answers would also diverge. While calculating Fleiss' kappa values and interpreting the results, we showed that students might be appropriate substitute for professionals, when simple aspects are questioned (e.g., source code size, also source code complexity level). In the case of investigating some complex aspects (e.g., a cohesiveness of a class), where day-to-day practice experience might help, students are not appropriate participants.

Based on the presented paper, we would encourage authors in the software quality research domain, to employ professional participants in their experiments. In cases, when simple answers are expected, students can also be appropriate. However, based on the approach, demonstrated by this paper, we would also encourage authors dealing with mixed participants, in terms of students and experienced professionals, to compare the student- and professional-based results in order to verify if the student-based data is valid.

As a side effect, we clearly showed in this paper that in the area of software quality research, experience (professionals) in addition to formal education (students) is crucial.

Author Contributions: Conceptualization, L.P., M.H. and T.B.; Data curation, T.B.; Formal analysis, L.P. and T.B.; Funding acquisition, M.H.; Investigation, T.B.; Methodology, L.P., M.H. and T.B.; Project administration, M.H.; Resources, M.H.; Software, L.P.; Validation, M.H. and T.B.; Visualization, L.P.; Writing—original draft, L.P. and T.B.; Writing—review and editing, L.P., M.H. and T.B. All authors have read and agreed to the published version of the manuscript.

Funding: The authors also acknowledge financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. SQALE. SQALE—Software Quality Assessment based on Lifecycle Expectations. 2020. Available online: <http://www.sqale.org> (accessed on 29 August 2020).
2. ISO/IEC 25040. *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—Evaluation Process*; ISO/IEC: Geneva, Switzerland, 2011.
3. Wohlin, C.; Runeson, P.; Hst, M.; Ohlsson, M.C.; Regnell, B.; Wessln, A. *Experimentation in Software Engineering*; Springer: Berlin, Germany, 2012.

4. Daun, M.; Hübscher, C.; Weyer, T. Controlled Experiments with Student Participants in Software Engineering: Preliminary Results from a Systematic Mapping Study. *arXiv* **2017**, arXiv:1708.04662.
5. Basili, V.R. The role of experimentation in software engineering: past, current, and future. In Proceedings of the IEEE 18th International Conference on Software Engineering, Berlin, Germany, 25–30 March 1996; pp. 442–449.
6. GIT-Scm. GIT. 2020. Available online: <https://git-scm.com> (accessed on 29 August 2020).
7. Falessi, D.; Juristo, N.; Wohlin, C.; Turhan, B.; Münch, J.; Jedlitschka, A.; Oivo, M. Empirical software engineering experts on the use of students and professionals in experiments. *Empir. Softw. Eng.* **2018**, *23*, 452–489. [[CrossRef](#)]
8. Feitelson, D.G. Using Students as Experimental Subjects in Software Engineering Research—A Review and Discussion of the Evidence. *arXiv* **2015**, arXiv:1512.08409.
9. Zhang, L.; Tian, J.H.; Jiang, J.; Liu, Y.J.; Pu, M.Y.; Yue, T. Empirical Research in Software Engineering—A Literature Survey. *J. Comput. Sci. Technol.* **2018**, *33*, 876–899. [[CrossRef](#)]
10. Sjöberg, D.I.K.; Hannay, J.E.; Hansen, O.; Kampenes, V.B.; Karahasanovic, A.; Liborg, N.; Rekdal, A.C. A survey of controlled experiments in software engineering. *IEEE Trans. Softw. Eng.* **2005**, *31*, 733–753. [[CrossRef](#)]
11. Feldt, R.; Zimmermann, T.; Bergersen, G.R.; Falessi, D.; Jedlitschka, A.; Juristo, N.; Münch, J.; Oivo, M.; Runeson, P.; Shepperd, M.; et al. Four commentaries on the use of students and professionals in empirical software engineering experiments. *Empir. Softw. Eng.* **2018**, *23*, 3801–3820. [[CrossRef](#)]
12. Boehm, B.W. Software Engineering Economics. *IEEE Trans. Softw. Eng.* **1984**, *10*, 4–21. [[CrossRef](#)]
13. Hughes, R.T. Expert judgement as an estimating method. *Inf. Softw. Technol.* **1996**, *38*, 67–75.
14. Institute, P.M. *Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 6th ed.; PMBOK® Guide, Project Management Institute: Newtown Square, PA, USA, 2017.
15. Steen, O. Practical knowledge and its importance for software product quality. *Inf. Softw. Technol.* **2007**, *49*, 625–636. [[CrossRef](#)]
16. Yamashita, A.; Moonen, L. Do code smells reflect important maintainability aspects? In Proceedings of the 2012 28th IEEE International Conference on Software Maintenance (ICSM), Trento, Italy, 23–28 September 2012; pp. 306–315.
17. Bigonha, M.A.S.; Ferreira, K.; Souza, P.; Sousa, B.; Januário, M.; Lima, D. The Usefulness of Software Metric Thresholds for Detection of Bad Smells and Fault Prediction. *Inf. Softw. Technol.* **2019**, *115*, 79–92. [[CrossRef](#)]
18. Oliveira, P.; Valente, M.T.; Bergel, A.; Serebrenik, A. Validating metric thresholds with developers: An early result. In Proceedings of the IEEE International Conference on Software Maintenance and Evolution, Bremen, Germany, 27 September–3 October 2015.
19. Rosqvist, T.; Koskela, M.; Harju, H. Software Quality Evaluation Based on Expert Judgement. *Softw. Qual. J.* **2003**, *11*, 39–55. [[CrossRef](#)]
20. Chen, Z.; Chen, L.; Ma, W.; Zhou, X.; Zhou, Y.; Xu, B. Understanding metric-based detectable smells in Python software: A comparative study. *Inf. Softw. Technol.* **2018**, *94*, 14–29. [[CrossRef](#)]
21. TIBCO Software. JasperReports® Library Source Code. 2019. Available online: <https://github.com/TIBCOSoftware/jasperreports/blob/master/jasperreports/src/net/sf/jasperreports/engine/export/JRTextExporter.java> (accessed on 5 August 2020).
22. Fleiss, J.L. Measuring nominal scale agreement among many raters. *Psychol. Bull.* **1971**, *76*, 378. [[CrossRef](#)]
23. Denham, B. Interrater Agreement Measures for Nominal and Ordinal Data. *Categ. Stat. Commun. Res.* **2017**, *232*–254. [[CrossRef](#)]
24. Landis, J.R.; Koch, G.G. The measurement of observer agreement for categorical data. *Biometrics* **1977**, *159*–174. [[CrossRef](#)]