"I know it when I see it" – Perceptions of Code Quality

ITiCSE'17 Working Group Report*

Jürgen Börstler
Blekinge Institute of Technology
Karlskrona, Sweden
jubo@acm.org

Harald Störrle
QAware GmbH
Munich, Germany
Harald.Stoerrle@qaware.de

Daniel Toll
Linnæus University
Kalmar, Sweden
daniel.toll@lnu.se

Jelle van Assema
University of Amsterdam
Amsterdam, The Netherlands
jelle.van.assema@gmail.com

Rodrigo Duran
Aalto University
Helsinki, Finland
rodrigo.duran@aalto.fi

Sara Hooshangi
George Washington University
Washington, DC, USA
shoosh@gwu.edu

Johan Jeuring
Utrecht University
Utrecht, The Netherlands
J.T.Jeuring@uu.nl

Hieke Keuning
Windesheim University of Applied Sciences
Zwolle, The Netherlands
hw.keuning@windesheim.nl

Carsten Kleiner
University of Applied Sciences & Arts Hannover
Hannover, Germany
carsten.kleiner@hs-hannover.de

Bonnie MacKellar
St John's University
Queens, NY, USA
mackellb@stjohns.edu

ABSTRACT

Context. Code quality is a key issue in software development. The ability to develop high quality software is therefore a key learning goal of computing programs. However, there are no universally accepted measures to assess the quality of code and current standards are considered weak. Furthermore, there are many facets to code quality. Defining and explaining the concept of code quality is therefore a challenge faced by many educators.
Objectives. In this working group, we investigated code quality as perceived by students, educators, and professional developers, in particular, the differences in their views of code quality and which quality aspects they consider as more or less important. Furthermore, we investigated their sources for information about code quality and its assessment.
Methods. We interviewed 34 students, educators and professional developers regarding their perceptions of code quality. For the interviews they brought along code from their own experience to discuss and exemplify code quality.
Results. There was no common definition of code quality among or within these groups. Quality was mostly described in terms of indicators that could measure an aspect of code quality. Among these indicators, readability was named most frequently by all groups. The groups showed significant differences in the sources they use for learning about code quality with education ranked lowest in all groups.
Conclusions. Code quality should be discussed more thoroughly in educational programs.

*Working group co-leaders: Jürgen Börstler, Harald Störrle, and Daniel Toll

CCS CONCEPTS

• **General and reference → Evaluation**; • **Social and professional topics → Quality assurance**; **Computer science education**; **Software engineering education**;

KEYWORDS

Code quality, programming.

ACM Reference Format:
Jürgen Börstler, Harald Störrle, Daniel Toll, Jelle van Assema, Rodrigo Duran, Sara Hooshangi, Johan Jeuring, Hieke Keuning, Carsten Kleiner, and Bonnie MacKellar. 2017. "I know it when I see it" – Perceptions of Code Quality: ITiCSE'17 Working Group Report. In *Proceedings of ITiCSE 2017 Working Group Reports (ITICSE-WGR'17).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3174781.3174785

1 INTRODUCTION

The ability to develop high quality software is a key learning goal of computing programs [17, 23]. However, there are no universal measures to assess the quality of code and current standards are considered weak [6]. Defining the concept of "good" code is therefore a challenge faced by many educators.

Programming textbooks typically emphasize the importance of quality, but rarely go beyond correctness and commenting in their treatment of quality. The value of thorough commenting beyond "self-documenting code" is debatable though [13], and there is little empirical evidence on the effects of comments on code quality [33]. An early and strong focus on thorough commenting (and tools like JavaDoc) might actually take time from more important quality issues. Quality aspects like understandability and maintainability get little attention, although they are of significant importance for students' professional careers.

Anything a professional software developer will do, be it development, testing or maintenance, will involve reading and understanding code that someone else has developed. Eventually, their code will also need to be read and understood by others. It is therefore important to better prepare students to write code that will be easy to understand by others.

Some educators use static analysis tools such as Checkstyle[1] or FindBugs[2] to support the assessment of programming assignments. For example, Keuning et al. [18] recently showed that many student programs in the (huge) Blackbox database contain quality issues as reported by PMD[3], and that students hardly ever fix an issue, even when they make use of analysis tools. These tools check a large range of potential quality issues, but it is not clear which of these tools are suitable or even appropriate in an educational context.

In this working group, we looked into the ways that students, educators, and developers perceive code quality, in order to investigate which quality aspects are seen as more or less important, and which sources of information regarding code quality issues are used by these groups.

2 RELATED WORK

In an ITiCSE'09 working group, we investigated the quality of object-oriented example programs from common introductory programming textbooks [7, 8]. Our results showed that the overall quality is not as high as one would expect, in particular regarding the exemplification of object-oriented properties.

Stegeman et al. [30, 31] analyzed normative statements about code quality from three popular texts on software development and compiled them into a set of 20 quality aspects. Furthermore, they interviewed three teachers based on a programming assignment, using this set of quality aspects. The quality aspects and the results from the interviews where then used to generate rubrics for the assessment of programming assignments.

Inspired by code inspections [14, 21], Hundhausen et al. proposed pedagogical code reviews in which student teams review each others' code using a checklist of "best practices" [15]. Their focus was on students' attitudes toward quality and the checklist items were at a high level. Their study did not show any significant effects of pedagogical code reviews on code quality, which is quite interesting, since research on inspections in general shows a positive impact on quality [3, 21].

Research shows that low level code features may affect code quality. Butler et al., for example, showed that flawed identifier names are associated with low code quality [10]. A recent study of 210 open-source Java projects regarding the effect of coding conventions showed that size, comments and indentation affected readability most [22]. Furthermore, recent research in program comprehension shows that misleading names are more problematic than meaningless names [2], but that specific one-letter variables still can convey meaning [5]. It has also been shown that low-level structural differences affect program understanding, for example that for-loops are significantly harder to understand than if-statements [1] and that "maintaining undisciplined annotations is more time consuming and error prone" than maintaining disciplined ones [24].

There is also a large body of work on software quality measurement [11], but there is little conclusive evidence on the relationship between the measurements and common software quality attributes [16]. Recent research actually questions the usefulness of many common software measures, since they no longer have any predictive power when controlled for program size [12].

Research also shows that educators do not have a sufficiently accurate understanding of the programming mistakes that students actually make. A recent large-scale study showed that "educators' estimates do not agree with one another or the student data" regarding which student mistakes were most common [9]. One conclusion from these results could be that many educators might base their teaching materials on invalid invalid assumptions about the issues that students have.

In our work, we therefore want to elicit first-hand data from students, educators and developers to better understand their perceptions of code quality.

3 METHOD

Our overall goal was to investigate the differences in perceptions of code quality held by students, educators, and developers.

3.1 Overall Process

Since this study involved 10 researchers from different countries and institutions, we developed guides and instructions to ensure that the same procedures were followed at each site. The materials included the following: an interview guide describing the study design and all tasks; a transcription guide summarizing guidelines for the interview transcription; a participant information sheet (and consent form) to ensure that all participants receive the same information about the study; and a detailed interview script with instructions regarding the phrasing of interview questions and suggestions for probing questions. While recruiting participants, we used a shared document to ensure in each participant group under study a good spread of interviewees (see Section 3.4 for details).

The actual interview contained three parts, where the main open question (Q4) was recorded and then transcribed. All data was then stored in a shared location. To avoid bias (e.g., by exposing data

[1]http://checkstyle.sourceforge.net
[2]http://findbugs.sourceforge.net
[3]http://pmd.github.io/

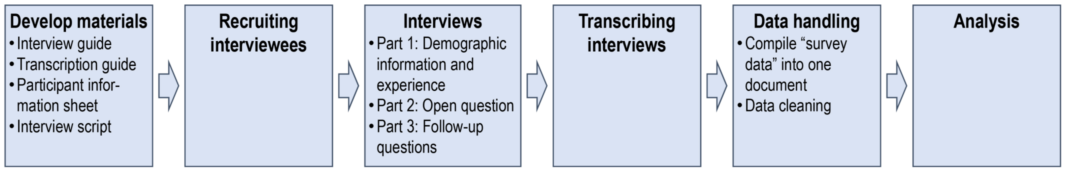

Figure 1: Overview over the study process.

already entered by others), the data was entered using forms to generate a shared spreadsheet later. After cleaning the data (see Section 3.5 for details), the group met in person for an analysis and discussion of the data.

The overall study process is summarized in Figure 1.

3.2 Research Questions

In this research, we followed an exploratory approach to get a better understanding of the perceptions of code quality. The research questions are therefore framed to be open and unbiased by preconceived hypotheses.

RQ1: How do participants define code quality?
RQ2: How do participants determine code quality?
RQ3: How do participants learn about code quality?
RQ4: Which tools do participants use for quality assurance?
RQ5: What are the differences in perceptions of code quality between students, educators, and developers?

We also collected information for a sixth research question (**RQ6**: What are the characteristics of actual example codes that are considered "good" or "bad"?), but this is not discussed further in the present report.

3.3 Data Collection

We used a detailed interview guide with predefined and scripted questions. When designing the questions, we took an exploratory approach to get a better understanding of the perceptions of code quality. Our goal was to explore perceptions of code quality, not to test preconceived hypotheses. The interview questions were therefore framed very carefully, so that they do not introduce bias or suggest certain answers (no leading questions).

Questions were tested using an initial pilot interview but this data was not used in the study. The interviews took 45–60 minutes and were either conduced in person or through video calls (using Google Hangouts, Skype, or Zoom).

The interview script contained 11 main questions of which one question was a semi-structured open question that was recorded and transcribed. The other questions were short free-text, numeric, or Likert-type questions. All Likert-type questions used a 7-item scale where only the end values were named explicitly (see Figure 2 for an example). The first 3 main questions (Q1–Q3) focused on demographics and the interviewee's overall experience and were filled in by the interviewer. Question Q4, the main part of the interview, was recorded and transcribed. The remaining 7 questions (Q5–Q11) were filled in directly by the interviewees. A list of all questions and subquestions can be found in Appendix A.

For Q4, we asked the interviewees to bring along code or code snippets to show us actual examples from their personal experience

"On a scale from *strongly disagree* to *strongly agree*, how much do you agree or disagree with the following statements regarding your personal experience related to source code quality."

- Code Quality is of high importance in my work/studies/teaching.
  Strongly disagree ☐ ☐ ☐ ☐ ☐ ☐ ☐ Strongly agree

Figure 2: Example of a Likert-type question (Q6a).

that exhibit characteristics that make the code sample "good" or "bad". If the interviewees brought examples in electronic form, we captured the screen to be able to connect the discussion to particular areas of the code.

For the transcription, we used Aine Humble's "Guide to Transcribing"[4] and adapted it for our purpose to ensure a more fluent transcript style. If possible, the interviews were held in the language the interviewees felt most comfortable in. All transcripts in other languages were finally translated to English.

3.4 Participant Recruiting/ Sampling

The main goal of our sampling was to recruit trustworthy key informants for a qualitative study. We targeted three groups of participants: students, educators, and developers. Educators needed to have at least a few years of experience with courses that deal with programming, software design, or software quality. For example, a teaching assistant who graded exercises in a programming course would not qualify as an educator. Developers needed to be people who actually deal with software development for a living, i.e. people who regularly read, write, test or review source code or low-level designs.

To recruit participants, we used a common information sheet that described basic details about the study. All researchers then collected basic information about potential participants (participant group, experience, gender, and country) on a common spreadsheet to make it easier to achieve a good spread among participants. To avoid participant group confusion (e.g., students qualifying as developers), we specifically asked about the level of teaching and professional programming experience.

3.5 Data Cleaning and Analysis

Before starting to analyze the interview results, the answers to the structured questions were cleaned up to improve the reliability of the results. Apart from fixing simple typos, the following clean up and harmonization was carried out:

- Spelling of country names and programming language names was harmonized.

[4]http://www.msvu.ca/site/media/msvu/TranscriptionGuide.pdf

- Units of measurement were harmonized for times (to years) and code length (to lines of code). In the case where ranges were given when individual numbers were expected, the average was used.
- Job titles of developers were harmonized, but potentially relevant differences were kept (e.g., *software architect* vs. *principal software architect*). Job titles primarily concerned with software development were uniformly classified as *software developer*.
- Interviewees who belong to more than one participant group were assigned to their primary group and their data was only counted for this group.

3.6 Coding of Open Questions

Categories were extracted inductively from the data using open coding. This was done on a an semantic level, thus we aimed not to go deeper than the surface themes, instead generic categories were selected to encompass the specific data. When new categories emerged or the definition or labels changed the whole dataset was categorized again using the new categories. Each data-entry was connected to one category. This was done until all data had been categorized and no new categories emerged and/or were changed. This is similar to the open coding step of grounded theory [20].

To analyze the data from open questions regarding definitions of code quality and factors/properties/indicators of quality (Q5 and Q8), we adopted the above described approach. One group of authors performed open coding on Q5, while another group did so, independently, on Q8. The initial open coding produced sets of labels that ranged from generic/abstract concepts to best practices and measurable examples.

As a next step, we merged the initial labels and categorized them into indicators and characteristics, where we defined an *indicator* as a measurable quality, for instance, *indentation*, *test case coverage*, or being free of bugs. A *characteristic* is then defined as a more abstract concept, that can be assessed or measured by means of indicators. The characteristic *readable*, for example, is based on indicators such as *formatting* or *adherence to naming conventions*.

3.7 Threats to Validity

Internal validity is concerned with the study design, in particular with the constructs (questions) used to answer the research questions. The questions should be suitable (i.e. questions will trigger useful and reliable information) and sufficient (i.e. the questions do not exclude essential aspects) to answer the questions.

The interview script contained closed and open questions. Question Q4 was intentionally left open and the interview guide encouraged interviewers to let the interviewees talk freely. We are therefore confident that all relevant aspects have been covered. In the question design, the answers to question Q9 (sources of information) might have been influenced by question Q6g ("I have learned ... from the Internet"). Internet-resources might therefore be over-represented in the answers for Q9.

All participants were informed in good time that they should bring code from their personal experience to the interview. This gave all interviewees time to think about code quality which might have influenced their answers. Since all participants were treated in the same way, we see no issues regarding a history threat here.

Participant selection is a threat in this study, since students were often self-selected, whereas educators and professional developers were more directly contacted by the researchers. In our student group, mature and dedicated students are likely to be over-represented.

External validity is concerned with the generalizability of the results. Since the sample size in this study is small, we cannot generalize the results to students, educators, and developers in general, in particular since there might be participant group confusion. Most students are at the end of their studies and about half of the students and the educators have some experience as developers (see Section 4.1). However, we would rather expect to get larger differences between groups with less group confusion.

4 RESULTS AND ANALYSIS

This section is organized as follows. After a summary of demographic data (4.1), we summarize data regarding the participants experience with reading, modifying and reviewing code (4.2) and their overall perceptions of code quality (4.3). In Subsection 4.4, we summarize how participants describe code quality. Data pertaining to sources of information and tools relevant for code quality are presented in Subsection 4.5 and 4.6, respectively. Group differences are touched in each of the subsections, therefore there is no separate subsection for their presentation.

4.1 Demographics

In total, we conducted 34 semi-structured interviews with students (12), educators (11), and developers (11) from 6 countries (The Netherlands: 9, Germany: 7, Sweden: 6, USA: 6, Finland: 5, UK: 1). Most of the interviewees were male (28). A detailed demographics of all participants can be found in Table 1. Of the students, the majority (9 of 12) were in their third to fifth year of studies or had just completed their studies. Furthermore, five students had some experience as a professional developer. Of the educators, about half (6 of 11) had some experience as a professional developer. Of the developers, 2 out of 11 had some experience as an educator.

4.2 Working with Code

We asked to which extent the participants read, modify or review other people's code or to which extent their code is read, modified or reviewed by other people (Q3-5). Figure 3 shows that a majority of participants read, modify and review other people's code, but only about half of them have their own code read, modified or reviewed by others. The differences between reading, modifying and reviewing code from other people and own code being read, modified and reviewed by other people are statistically significant at $\alpha < 0.05$ ($\chi^2 = 6.3143; p = 0.043$).[5]

When divided into subgroups (see Figure 4), we can see statistically significant differences according to a Fisher's exact test ($p < 0.01$). Almost all developers read, modify and review code from other people, whereas only about half of the students do so,

[5]For the statistical tests (χ^2 and Fischer's exact test), we grouped all negative answers into one group, the neutral answers into one group, and the positive answers into one group.

Table 1: All participants in the study. gender (Q1), country (Q2), and group (Q3).

Participant	Gender	Country	Group
RD3	Female	Finland	Student
RD1	Male	Finland	Student
RD2	Male	Finland	Student
CK3	Male	Germany	Student
JB5	Female	Sweden	Student
JB4	Male	Sweden	Student
HK3	Male	The Netherlands	Student
JA2	Male	The Netherlands	Student
JA3	Male	The Netherlands	Student
BM3	Male	USA	Student
SH1	Male	USA	Student
SH2	Male	USA	Student
RD4	Male	Finland	Educator
RD5	Male	Finland	Educator
CK1	Male	Germany	Educator
DT3	Female	Sweden	Educator
DT1	Male	Sweden	Educator
HK1	Male	The Netherlands	Educator
JA4	Male	The Netherlands	Educator
JJ2	Male	The Netherlands	Educator
JB1	Male	United Kingdom	Educator
BM2	Female	USA	Educator
SH3	Female	USA	Educator
CK2	Male	Germany	Developer
HS1	Male	Germany	Developer
HS2	Male	Germany	Developer
HS3	Male	Germany	Developer
HS4	Male	Germany	Developer
DT5	Female	Sweden	Developer
DT2	Male	Sweden	Developer
HK2	Male	The Netherlands	Developer
JJ1	Male	The Netherlands	Developer
JJ3	Male	The Netherlands	Developer
BM4	Male	USA	Developer

with educators lying in between. A similar pattern can be seen for reading and modifying other people's code. Regarding code that is reviewed by other people, the pattern is different. The code written by educators is reviewed by other people to a much lesser extent than the code written by students or developers.

4.3 Overall Perceptions of Code Quality

The majority of participants strongly agreed that code quality is of high importance in their work, studies, or teaching (Q6a; mean: 6 on a scale from 1 to 7)[6]. Although a majority of participants also agreed that they can easily tell good from bad code (Q6b; mean: 5.1), they agreed less strongly with the statement that they know how to measure code quality (Q6d; mean: 4.6). The differences

[6]The scale reaches from 1 (strongly disagree) to 7 (strongly agree), see Figure 2. A mean below 4 would therefore correspond to disagreement and a mean above 4 to agreement.

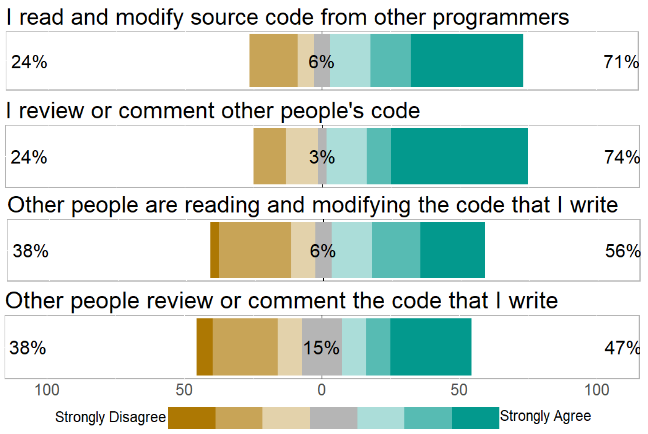

Figure 3: Experiences of all participants (n=34) regarding interaction with other developers' code (Q3-5a–Q3-5d).

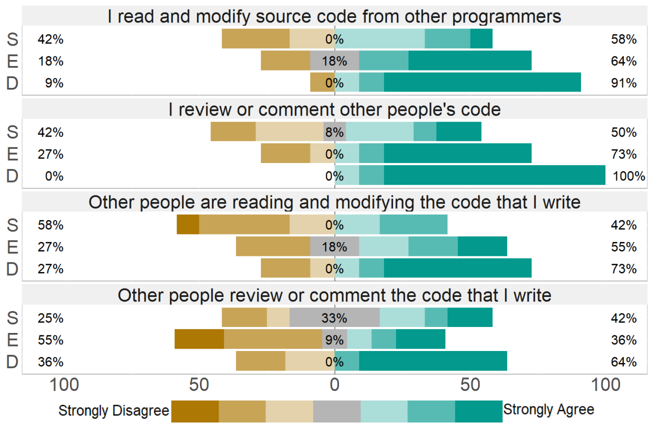

Figure 4: Experiences of participants regarding interaction with other developers' code (Q3-5a–Q3-5d) by subgroups (S=students, E=educators, D=developers).

between Q6b and Q6d are not statistically significant at $\alpha < 0.05$ ($\chi^2 = 1.5814; p = 0.454$) and roughly the same for all groups. The results in Q6b, and Q6d varied among the three groups with students agreeing the least in being able to tell good from bad code (Q6b; mean: 4.2) and students disagree slightly to know how to measure code quality (Q6d; mean: 3.6). Developers most strongly agreed to both questions (Q6b; mean: 5.7 and Q6d; mean: 5.3). This can also be seen in Figure 5.

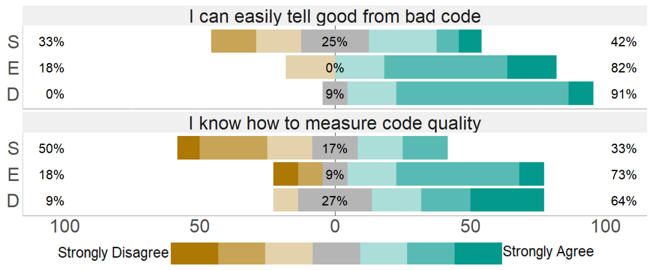

Figure 5: Participants' level of agreement to Q6b (top) and Q6d (bottom) by subgroups (S=students, E=educators, D=developers).

4.4 Describing Code Quality

4.4.1 Terms Used to Describe Code Quality. We asked the participants to define code quality and describe features and properties that best represent code quality (Q5). We also asked them to list the three topmost quality factors or indicators of high quality code (Q8). The answers tended to be a mix of abstract characteristics and indicators with unclear definitions of the terms. Following a grounded theory analysis (see Subsection 3.6), the complete set of characteristics and indicators generated from the coded answers is shown in Table 2.

Table 2: Categorization of terms used to define code quality (Q5) and indicators/ characteristics of code quality (Q8).

Category	Terms used to describe code quality
Readability	readable, no useless code, brevity/ conciseness, formatting/ layout, style, indentation, naming convention
Structure	well structured, modular, cohesion, low coupling, no duplication, decomposition
Testability	testable, test coverage, automated tests
Dynamic behavior	robust, good performance, secure
Comprehensibility	understandable, clear purpose
Correctness	runnable/ free of bugs, language choice, functionally correct (meeting business requirements)
Documentation	documented, commented
Maintainability	maintainable, adaptable, reusable, used by others, interoperable, portable
Miscellaneous	license, suitable data structure, metrics/ measurements

4.4.2 Definitions of Code Quality. We counted the number of occurrences of a given term in terms of the total of participants to analyze definitions of code quality based on the ground data provided by the interviewees (in Q5) and the categorization of terms described in Subsection 4.4.1.

In general, our population converged to two categories: *readability* with 82% of overall frequency and *structure* followed as the second most cited term with 62%. For the other terms there was no clear convergence and a much lower frequency, as Figure 6 shows.

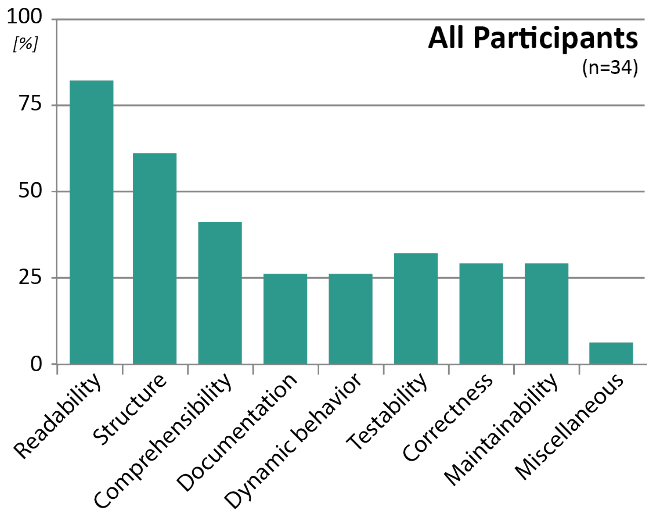

Figure 6: Terms used to define code quality (Q5). The y-axis shows the percentage of participants that cited terms in a given category.

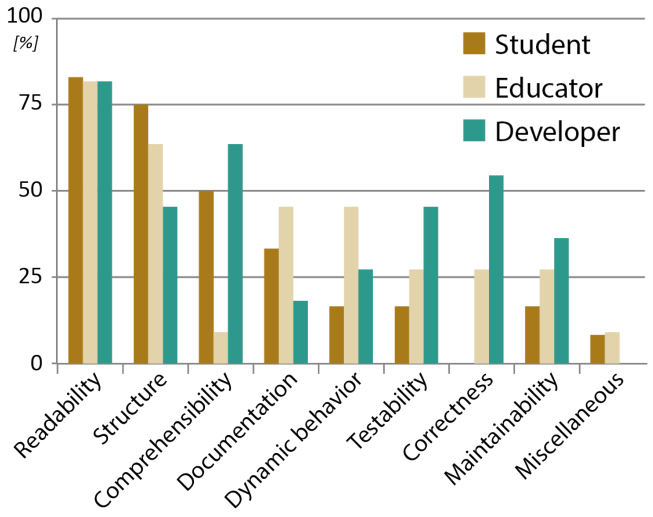

Figure 7: Terms used to define code quality by subgroup (Q5, from top to bottom): students, educators, and developers. The y-axis shows the percentage of participants that cited terms in a given category.

However, when looking into the subgroups of the interview data, each subgroup showed distinct differences in frequencies of mentioned terms, beyond the common convergence toward *readability* as the top category (see Figure 7).

4.4.3 Indicators of Code Quality. We analyzed in more detail the priority of the cited terms using data from Q8, which asks participants to name their three topmost quality factors or indicators of high quality code. Figure 8 shows the frequencies of terms mentioned as a first, second or third priority for all participants and split by subgroups. In total, the students clearly mentioned terms related to *readability* the most, followed by *comprehensibility* and then *maintainability*. For their first choice, the students‘ answers are more evenly distributed between several categories. Some concrete examples of indicators that we found in the student responses were *"commented-out code can indicate bad code"*, *"the indent of the code is not too deep, more than 4 tabs"*, and *"good balance in use of comments"*.

In total, the educators most often mention terms belonging to the *structure* category (e.g., *"conformance to general design patterns and architectures"*), followed by terms belonging to the *readability* and *correctness* categories. As a first choice *readability* is most often mentioned.

In total, the developers most frequently mention the *readability* category, which is also their first choice most often, followed by *comprehensibility* (such as *"code can be understood and/or used in isolation"*), and then *correctness* and *testability* (such as *"code is thoroughly tested with unit tests, integration tests"*).

Terms related to *structure* are hardly mentioned by developers, while educators consider *structure* terms to be very important to indicate high quality code. Terms regarding *testability* are only mentioned by developers. Students do not mention *correctness* at all, while developers and educators do. Educators do not name any terms related to *dynamic behavior* at all.

4.5 Learning about Code Quality

We used several questions to elicit respondents' sources of information regarding code quality. For example, in Q6e–g, we asked how much they learned during their education or from colleagues and the Internet respectively. Overall, education was ranked lowest (by all groups), and colleagues were ranked highest, but there were highly significant differences at $\alpha < 0.01$ between the three groups of respondents ($\chi^2 = 22.276; p = 0.0002$). Students valued the Internet highest, whereas educators and developers valued colleagues highest. Figure 9 shows how the sub-populations rate the importance of formal education, colleagues, and the Internet as sources for learning about code quality.

In question Q9, we asked participants about the most useful sources of information about code quality. All in all, participants mentioned over 30 different sources of information. They can be grouped into the following seven categories (sorted by decreasing frequency of mentioning, see Figure 10).

- **Internet** refers to all online sources, mostly StackOverflow and GitHub, but also mentioning reddit, W3Schools, and Google.
- **Colleagues** were named as important sources by many participants, in particular experienced colleagues. We include in this category mentions of code reviews, and *"peer students"*.
- **Textbooks** have been mentioned as information sources in general, where all printed material has been collected in this category. The book "Clean Code" [25] has been mentioned

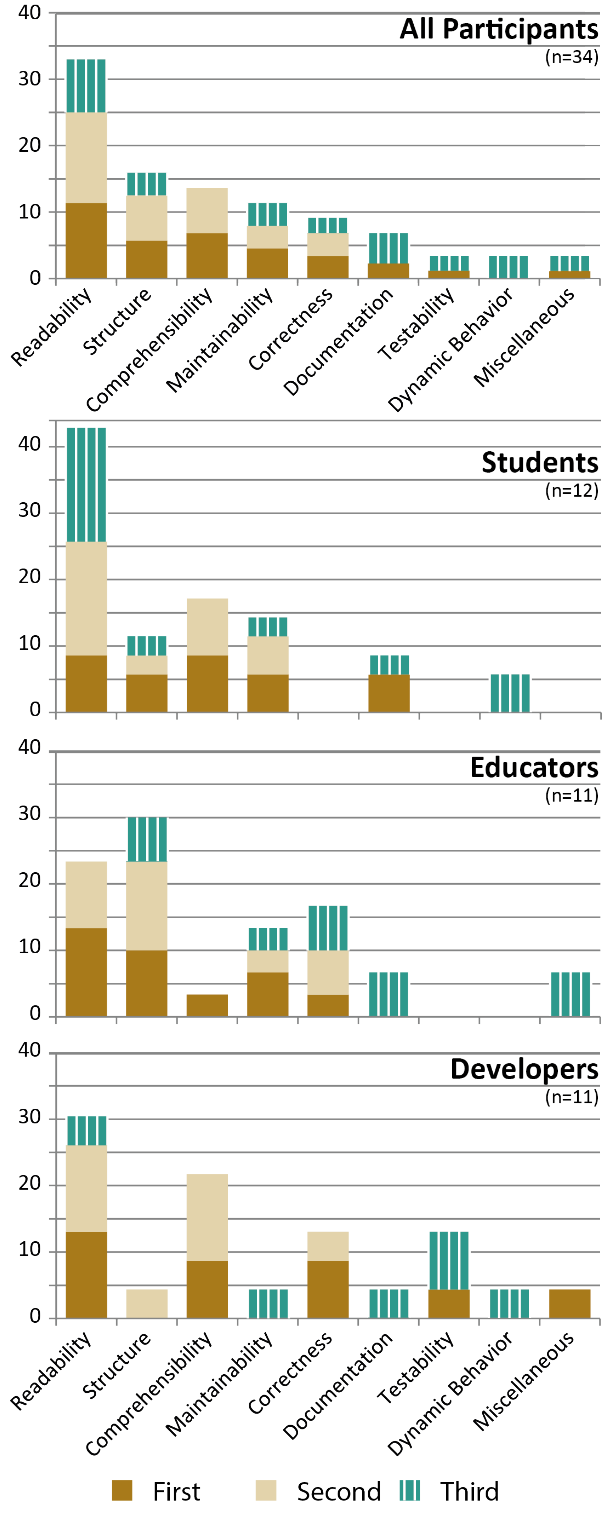

Figure 8: Categories of the three topmost quality factors or indicators of high quality code (Q8) for all participants (top) and by subgroup (from top to bottom): students, educators, and developers. The y-axis shows the percentage of terms of all terms cited by participants that fell in a given category. E.g., about 30% of all terms cited by developers were in category Readability. Percentages in each graph add up to 100%.

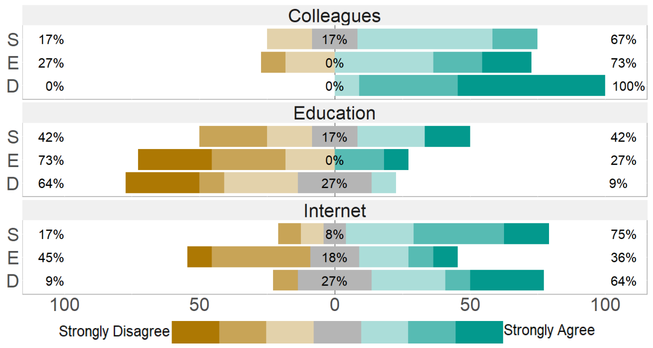

Figure 9: Importance of colleagues (top), education (middle) and the Internet (bottom) for learning about code quality (Q6e–g) by subgroups (S=students, E=educators, D=developers).

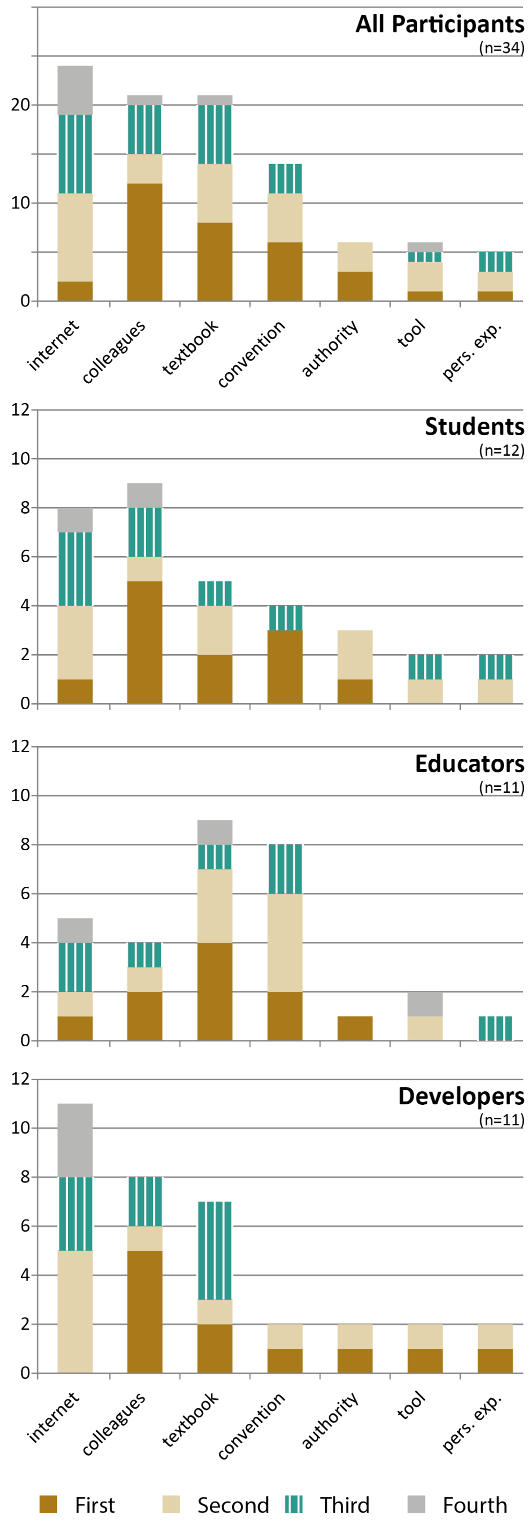

Figure 10: Most useful information sources on code quality (Q9) for all participants (top) and by subgroup (from top to bottom): students, educators, and developers. The y-axis shows the total mentionings per resource category aggregated by rank from first (bottom of each bar) to fourth (top).

the most by far, other books can be found in the references section ([4, 27, 28]). Some participants explicitly mentioned textbooks as poor sources that they would not consider reliable or helpful. (*"'life support literature' - not very helpful"*).

- **Conventions** include language specific coding standards, organizational and project specific conventions, and also lore.
- **Authorities** refers to well known figures that might appear on technology conferences, but also academic luminaries such as E. W. Dijkstra, and academic teachers.
- **Tools** includes recommender systems like code completion and refactoring proposals, but also tool documentation, and on-line help for tools. Note that tools are not necessarily automated tools but may also be best practices or methodologies.
- **Personal experience** refers to lessons learned from reading own and foreign code, and past projects.

4.6 Tool Support for Code Quality

We also analyzed which tools were suggested by participants to support code quality assurance or control. In Q10, we asked for the first to third most useful tool as well as for other useful tools.

As a first step we aggregated all tools, regardless of their priority, into general tools for software development (like IDEs), specific tools or add-ons for code quality in particular and a third category for suggestions that didn't fit any of these two tool categories, e.g., *planning before coding* or *search in existing libraries*. Our results show that 38% of the tools are regular software development tools and 56% are specific code quality tools. As shown in Figure 11, the largest difference between subgroups is that developers to a much higher degree (71%) mention specific code quality tools.

Responses to this question have been harmonized and then grouped by primary purpose, irrespective of the particular tool name mentioned. For instance, all development environments were classified as *IDE*. In cases where tools can be used for different purposes, the primary purpose as stated by the vendor has been used. For example, JUnit can be used for unit testing and also in continuous integration scenarios. We grouped it into testing support,

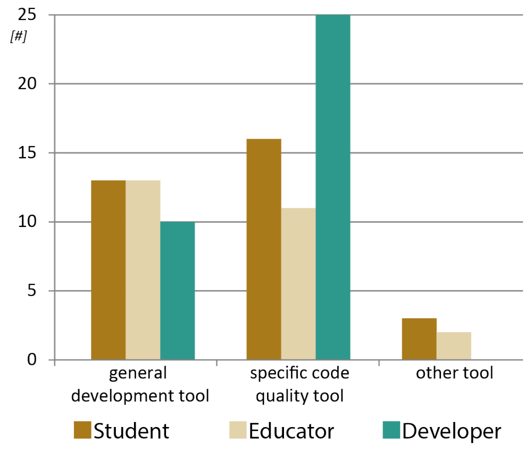

Figure 11: Distribution of general versus specific tools for code quality by subgroup (derived from Q10).

because it can be used independently of continuous integration as well. Jenkins, on the other hand, is primarily a continuous integration tool, thus we grouped it there.

In Figure 12, we list tool groups with at least 3 overall mentionings separately. In category *misc*, we grouped all tool groups and other responses with at most 2 overall mentionings.

As shown in Figure 12, IDEs are mentioned most often as an important code quality tool in total and often as topmost important tool. Code reviews are mentioned as the topmost important tool almost as often as IDEs in all subgroups. Version control is mentioned the least frequently and with low importance by all subgroups.

There are also substantial differences between the subgroups. Educators are the only group where category *misc* was mentioned as topmost import. In fact educators have this category as the most frequent topmost category and second largest category in total. For developers, continuous integration is the largest group, whereas it is among the smallest for students and educators.

5 DISCUSSION

In this section, we discuss the results and analysis from the previous section in more detail. The organization into subsections follows the organization of the previous section.

5.1 Demographics

Since about half of our students and educators have some experience from professional software development, there was a risk for obtaining only small differences between the groups as the distinction might have been vague. Surprisingly, this was not really the case. Still we assume that the group differences would have been even more pronounced with a clearer distinction between group experiences with respect to professional software development.

5.2 Working with Code

Our results in Section 4.2 show that participants, in general, are working to a significantly higher degree with other people's code as compared to their own code being read, modified or reviewed by others. This could be expected, since even small programs involve

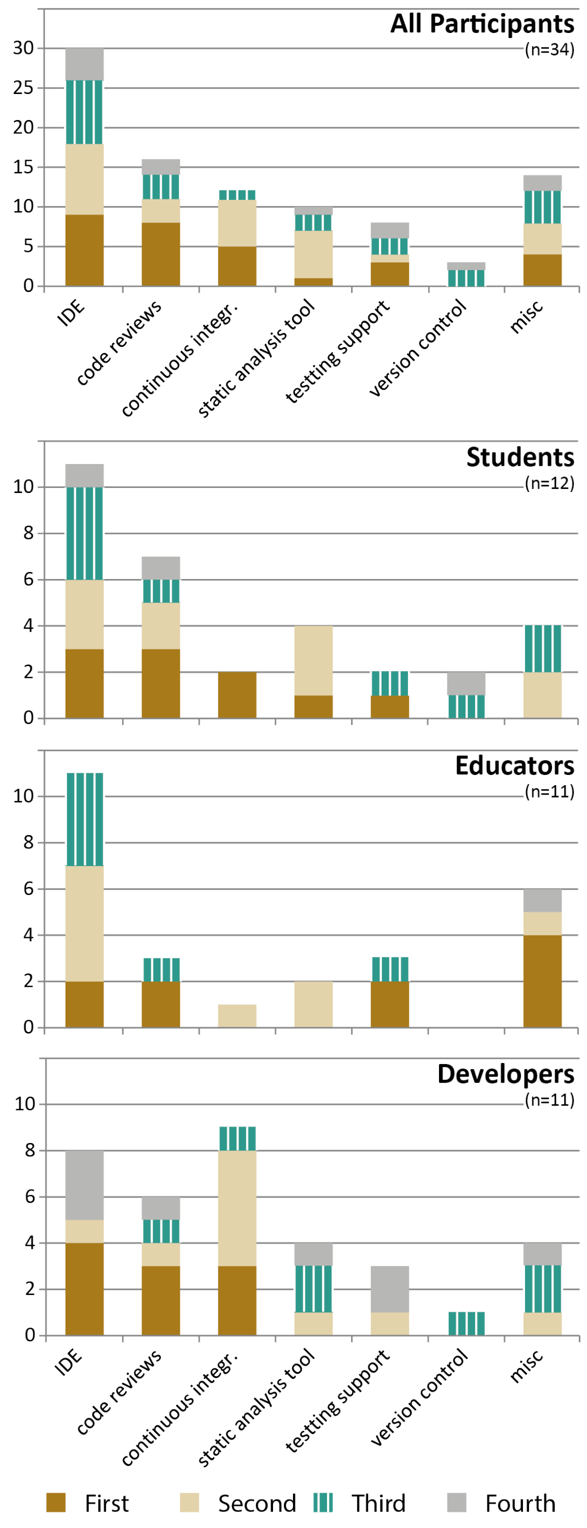

Figure 12: Most useful tools for improving code quality (Q10) for all participants (top) and by subgroup (from top to bottom): students, educators, and developers. The y-axis shows the mentionings per tool category aggregated by rank from first (bottom of each bar) to fourth (top).

the usage of code written by someone else. When looking at the answers for students, educators and developers, we can see large differences though.

Students are not reading, modifying, and reviewing other people's code as much as developers or educators, but still 50–58% agree that they do so (see Figure 4). Since most of our students are advanced students, they might have been exposed to software development projects that involve reading, modifying, and reviewing code of fellow team members. Student's code also is read and modified less than overall, while it is being reviewed and commented similarly to the overall number. It is still surprising, though, that only 42% of the students agree that their code is reviewed and commented by others. Since students write code for assessment, we would expect this percentage to be higher.

Since students also rate the importance of their education for learning about code quality relatively low (see Figure 9), this might be an indication that students do not get the support they need. It could be that students' code is not actually reviewed for assessment, for example when they just need to hand in code that passes certain tests, no matter what the code looks like. It could also be that the students for some reason do not benefit from the reviews they get. For their learning and motivation it would be very important to get detailed feedback beyond "your submission passed all tests". In our interviews, students frequently asked for more and more specific feedback about what was good or bad in their code.

Using static analysis tools, like Checkstyle, FindBugs or PMD, could help students with such questions, but would require that the students are taught how to use these tools and, in particular, how to interpret the results. Automating feedback has not yet been very successful [19, 32]. Students need feedback, not only to write better code, but also to learn how to give feedback to others. Peer reviewing code could be a useful tool to teach students ways to assess code and to give constructive feedback [29]. Interestingly, students mention code reviews as the second most important tool for improving code quality (see Figure 12), although only 42% of them agree that their code is reviewed and commented by others.

Educators work on other people's code more than students but less so than developers. They are also reading and modifying other people's code more than students but less so than developers. This is in-line with an educator's primary duties and could be expected. Only 36% of the educators agree that their code is reviewed or commented by others, which is slightly lower than for students (42%) and much lower than for developers (64%). To be able to provide students with high quality examples, code used in education should be of high quality. Earlier research has shown that the code quality of textbook examples has quite some potential for improvement (see, e.g., [8]). Reviewing and commenting the code written by educators to a higher degree might help to improve this situation.

Developers work on other people's code to a higher degree than students and educators. Most of the developers actually strongly agreed that other people are working with their code. The few developers who disagreed might be developers in leading roles such as principal software architect or technical chief designer, who write little code by themselves, but have quality assurance (involving code reviews) of other developer's code as one of their main tasks.

5.3 Overall Perceptions of Code Quality

Our results show that most participants strongly agree that code quality is important for their work, studies or teaching. Educators and developers are also very confident in being able to tell good code from bad code (82% and 91% agreement, respectively, see Figure 5). Most educators and developers agree that they know how to measure code quality, whereas more than half of the students do not agree with this statement.

There is a range of potential explanations for students' uncertainty. Students' overall lower agreement might be due to their lower interaction with peers during their education (see Figure 4). Knowledge about what is regarded as good code requires experience which can be built by interacting with others through code.

As discussed in the previous section, students read, modify, and review/ comment other people's code to a much lower degree than educators and developers. Furthermore, for students, colleagues (i.e. fellow students) are the overall first choice for learning about code quality (see Section 4.5). It therefore seems that students might have too little exposure to aspects of code quality and examples of high quality code during their education. Nevertheless, students more often than other groups report to have learned a lot about code quality in their education (see Figure 9). This might either be due to an improved importance of teaching code quality in education in recent years or because educators and developers value their practical experiences regarding code quality much higher than what they learned during their education.

5.4 Describing Code Quality

Our results show that *readability* is the dominant category of terms for defining code quality (Q5; *"The major feature of code-quality is that it should be readable like a natural language."* – student). It is also the dominant category for factors or indicators of high quality code (Q8) for all participants and for all groups. Beyond readability, there are large differences between groups.

Students are focused on the minutiae of the code. Our data shows that *readability* (83%) and *structure* (75%) are categories that clearly define code quality according to students (Q5, see Figure 7). *Code comprehension* also plays a major role with 45% of occurrences. Students seem to have a narrow view of code quality. Only 4 of our 9 categories of terms are used by 25% or more of the students. Terms from the categories *dynamic behavior*, *testability*, *maintainability* or *miscellaneous* are used infrequently. *Correctness* wasn't used at all. A reason for that might be that they have little or no exposure to these issues, because students typically do not write very large programs, programs that are used by other people, or programs that life for more that a study term.

When looking at the topmost factors or indicators of high quality code (Q8, see Figure 8), the picture is even more pronounced. About 43% of all factors named by students fell in category *readability*.

Educators show a broader view when defining code quality with a wider range of categories that are used by 25% or more of the educators (Q5, see Figure 7). Similarly to students, *readability* (82%) and *structure* (64%) are the categories that are mentioned most frequently. *Documentation* and *dynamic behavior* were mentioned by 45%, which is substantially higher than for students or developers. The reason for this broad view might be attributed to the range of courses that the educators teach, their area of research, and how much prior industry experience they have. There might be a higher diversity in the backgrounds, work tasks and interests of computer science educators than for the other groups.

Interestingly, educators use slightly different categories for the topmost factors or indicators of high quality code (Q8, see Figure 8) than for defining code quality. *Structure* and *readability* are still the most important factors, but in reverse order. *Documentation*, *dynamic behavior*, and *testability* are barely mentioned, although they are used frequently in defining code quality.

Professional developers also agreed on the importance of *readability* (82%). *Comprehensibility* was also used frequently (64%), which is higher than for the other groups (Q5, see Figure 7). Developers, like educators show a broad view of code quality. Only 2 categories, *documentation* and *miscellaneous*, were used by less than 25%. The developers appear to be more concerned with all levels of software development, particularly when compared to students. For example, they much more frequently described code quality in terms of *correctness* and *testability*. This is not too surprising, since professional developers have to focus on many software concerns beyond actual coding. In particular, developers have very likely more experience with the overall life-cycle of software systems and therefore seem more concerned with software system maintenance over time as compared to students and educators.

When looking at the topmost factors or indicators of high quality code (Q8, see Figure 8), developers show the most consistent picture. The answers to Q5 and to Q8 are very similar.

Summarizing, we can observe that although questions Q5 (defining code quality) and Q8 (topmost indicators of quality) are quite similar, the answers to Q5 and Q8 show a few differences. Readability, structure and comprehensibility are used most frequently in the answers to Q5 as well as to Q8. Maintainability was used more prominently in the answers to Q8 than to Q5. Our results also show that students and educators use fewer categories of terms in their answers to Q8 than to Q5. It seems that students and educators need more concepts to define code quality than to describe the topmost indicators of code quality. We hope that a detailed analysis of our qualitative analysis (i.e. the transcripts of Q4 and the actual code examples) will give us more insights into this issue.

When comparing the terms mentioned most in our study with common software quality models or standards, such as ISO/IEC 9126 or ISO 25010 [26], we observe that the top categories in our study are not even mentioned there. While the main focus of the named standards is more the software system as a product than the internals, it is still not quite clear how and where, e.g., readability and structure fit into them. This might make it difficult for programmers to relate their work to those standards in practice. This might also show a need for a specification of software code quality standards that reflect issues in the software developers' daily work.

5.5 Learning about Code Quality

In this section we discuss the results from Q6e–g and Q9 as reported in Section 4.5. From Figure 9 it is very obvious that education is the least favored source for learning about code quality in all subpopulations. It is especially true, though, for the developers among which colleagues are extremely important. Overall the importance of colleagues followed by textbooks is also reflected in Figure 10 where those are named most often as most useful source of information. Internet while being named most often overall is only named as second, third or other useful source by most participants. This shows that Internet can only complement information about code quality obtained from other sources. Probably this is due to many examples of bad code which are also shown there.

Looking at sub-populations and priority of mentioning, interesting differences emerge (see Figure 10). While developers mention Internet most often overall, they mention colleagues most often as their preferred source. None of the developers participating in our study mentioned Internet as the first source they would consult. Students mention colleagues most often in total as well as their first choice, but they mention the Internet slightly less often. This is a difference to the results from Figure 9, but only small in number so this does not seem to deserve further investigation.

In contrast, educators mention textbooks and conventions most often as their primary source of information, both in overall frequency and preference. It seems that educators do rely more on well developed, designed and objective sources as opposed to more dynamic or subjective sources. We hypothesize that this might be due to the nature of academic work that requires to justify arguments by references, which is not readily available for colleague's opinions. This is in line with the fact that most educators are also working in research where a proven quality of information sources is important.

Even if various Internet sources are mentioned by all developers, it is not considered as reliable as asking a colleague. We hypothesize that the conventions might not have been named that much by developers because some of them refer to the book "Clean Code" [25]. This book basically is a coding standard/convention while still being counted as a book. Developers are mentioning their IDEs as their first choice of tools for increasing code quality (see Figure 12), thus conventions might be considered encoded into these tools (for example as linter's), or that these conventions are enforced by colleagues during code reviews and are not considered a source of information themselves.

The student subgroup mentions sources of information very similarly to the developer subgroup, it is clear that students use Internet and their student colleagues to a much higher degree than learning about quality from their teachers (authority).

5.6 Tool support for code quality

Our results in Section 4.6 show that many of the tools mentioned are not tools specifically used for ensuring code quality but are developer tools like IDE and version control. However, developers mention to a much higher degree specific tools used to raise code quality (see Figure 11). We assume that only professional developers are aware of many of the code quality specific tools because these tools are not frequently used in education settings, i.e. by students end educators.

Combined from all three groups the three most commonly mentioned tools mentioned were IDEs, code reviews and tools for continuous integration. If we divide the results into the subpopulations (see Figure 12), we see that the student group is primarily concerned with developer support through IDEs and also with code reviews.

This seems somewhat surprising as Figure 4 showed that students do not frequently do code reviews, yet they seem to be aware of their importance for code quality.

Professional software development tools such as continuous integration and static analysis are less mentioned by the students, probably due to their lack of experience with professional software development processes. The students who mentioned these are part-time software developers in addition to their studies, so that is why they have been exposed to these type of tools already. Also mentioned by students is version control which seems to be natural to the other groups and thus probably was not worth mentioning for them.

The results for educators reveal that educators primarily mentioned IDEs and, less frequently, code review and testing support. The comparatively large *others* section mainly consists of things that a single educator based on his personal perspective considered important. As there is no common agreement, they need not be investigated further.

In contrast to the previous results, developers considered continuous integration tools very important, mentioning them most. Of interest is the high number of *Second* mentions, meaning continuous integration tools are very important in that they complement other tools for high quality. IDEs and code reviews also received high numbers of references, particularly as *First*. Also mentioned, but neither often nor at a high priority are static analysis tools and testing support tools. That seems a little bit surprising, particularly given the comparatively high number of references to these by the student subgroup. It seems that student expectations differ from professional practice in this aspect. Our hypothesis is that since students may have been graded in part based on static analysis tools, they may be used to them in that context.

6 SUMMARY

We have studied code quality as perceived by students, educators, and practitioners in several European countries and the US. We have collected quantitative and qualitative data but have so far only analyzed the quantitative part. The results have been rich and varied already, leading to several insights. In the following subsections, we summarize our initial findings regarding each of the research questions from Subsection 3.2.

6.1 RQ1: How do participants define code quality?

Our data did not reveal a single common definition of code quality among participants or participant groups. The definitions used the following nine “attributes” to define code quality, in the following order of importance (in terms of how often they were used): *readability*, *structure*, *comprehensibility*, *documentation*, *dynamic behavior*, *testability*, *correctness*, *maintainability*, and *miscellaneous*. All groups used *readability* as the most frequent attribute in their definitions and either *structure* (students and educators) or *comprehensibility* (developers) as the runner up. For educators *comprehensibility* ranked last of all nine attributes, whereas it was among the top-3 for students and developers. *Correctness* was named third by developers, but not much by educators and not at all by students. *Documentation* was barely used by developers, but quite frequently by students and educators.

6.2 RQ2: How do participants determine code quality?

Code quality i determined pretty much by the same factors that are used to define code quality (see RQ1). *Readability* is mentioned as an important quality factor by all groups, but not always the most important one. Educators name *structure* as the most important quality factor. Overall, we saw that fewer “attributes” were used to describe the most important quality factors as to define code quality. Even here, we saw several group differences. Whereas developers use all nine attributes (see Subsection 6.1, above for the full list), students do not mention *testability*, *correctness*, and *miscellaneous*, and educators do not mention *testability* and *dynamic behavior*.

6.3 RQ3: How do participants learn about code quality?

Regarding the sources for learning about code quality, the groups showed significant differences. However, all groups ranked their education as the least important source of information regarding code quality. Colleagues and the Internet are the highest ranked sources among students and developers, while educators seek knowledge in textbooks and in conventions to a higher degree than the other groups. Students and developers rely on colleagues to a high degree, but also on the Internet, although the latter is not their first choice.

6.4 RQ4: Which tools do the participants use for quality assurance?

The most commonly named tools for quality assurance were IDEs (for students and educators) and tools for continuous delivery (for developers with IDEs as runners up). Students and developers mention a more diverse tool-set than educators and focus on code reviews to a higher degree than educators. Educators, on the other hand, to a high degree name very specific tools as their first choice. Less than a third of the participants use static analysis tools for code quality assessment/ improvement, and only a single student named it as the first tool of choice.

6.5 RQ5: What are the differences in perceptions of code quality between students, educators, and developers?

We have seen many similarities between the groups (see subsections above), but also a few unexpected differences, which we briefly summarize below. In many aspects it seems that the perceptions of students are more similar to those of developers than to those of educators.

- Students are much less confident in telling good from bad code and to measure code quality.
- Students and developers mention colleagues and Internet as their main sources for learning about code quality to a higher degree than educators.
- Students and developers use a broader range of (types of) tools than educators.

- Students and developers mention code reviews as a tool to improve quality to a much higher degree than Educators.
- Students agree to a higher degree and developers to a much higher degree than educators with the statement that their code is reviewed or commented by others.

7 CONCLUSIONS AND FUTURE WORK

We collected quantitative and qualitative data about students', educators' and developers' perceptions of code quality. Up to this point, we have analyzed and discussed the quantitative data from 34 semi-structured interviews carried out in several European countries and the US.

The perceptions on code quality can be grouped into nine categories or themes: readability, structure, comprehensibility, maintainability, correctness, documentation, dynamic behavior, testability, correctness, maintainability, and miscellaneous. Since there are no empirically based categories for (perceptions of) code quality, our categories would be a good starting point for a large-scale survey.

Our results show that readability is perceived as a key characteristic of high quality code. It is, however, not quite clear whether all participants understand readability in the same way. A more detailed analysis of our qualitative data will give us more insight into this issue.

Our results also show that students' perceive that their code is reviewed and commented to a low degree, which is surprising since students typically write code for the purpose of assessment. They also get less information regarding code quality from their education than from other students or the Internet. This might have significant implications for their skill acquisition as well as their motivation. Giving students more exposure to issues regarding code quality as part of their education, e.g., by reviewing other people's code or tools that can be used for code quality assessment could solve some of these issues.

In many aspects it seems that the perceptions of students are more similar to those of developers than to those of educators. This is good and bad. Good in the sense that students develop in the right direction, but bad in the sense that education could be too far from the actual needs of students.

For our future work, we will analyze the qualitative data from our interviews (Q4). Furthermore, we have asked participants to bring along code examples to discuss during the interviews. These examples were used to point out code elements or sections that the participants considered of good or bad quality. In our future work, we will analyze those artifacts and cross-reference them to the interview transcripts.

We also plan to derive a body of representative examples of code, that is tagged with quality verdicts, and justifications for that verdict. Such a dataset will be helpful in creating a benchmark for an objective and comprehensive assessment of code quality. This will be helpful for educators and researchers as well. Educators can resort to the benchmark to find examples of "good" and "bad" code for their education. Researchers can use the benchmark as a "gold standard" for further research on issues related to code quality.

ACKNOWLEDGEMENTS

A part of this work is funded by the Netherlands Organization for Scientific Research (NWO), grant number 023.005.063.

REFERENCES

[1] Shulamyt Ajami, Yonatan Woodbridge, and Dror G Feitelson. 2017. Syntax, predicates, idioms: what really affects code complexity?. In *Proceedings of the 25th International Conference on Program Comprehension.* 66–76.

[2] Eran Avidan and Dror G Feitelson. 2017. Effects of variable names on comprehension an empirical study. In *Proceedings of the 25th International Conference on Program Comprehension.* 55–65.

[3] Gabriele Bavota and Barbara Russo. 2015. Four eyes are better than two: On the impact of code reviews on software quality. In *Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution.* 81–90.

[4] Kent Beck. 2002. *Test Driven Development: By Example.* Addison-Wesley Professional.

[5] Gal Beniamini, Sarah Gingichashvili, Alon Klein Orbach, and Dror G Feitelson. 2017. Meaningful identifier names: the case of single-letter variables. In *Proceedings of the 25th International Conference on Program Comprehension.* 45–54.

[6] Barry Boehm. 2016. Improving and balancing software qualities. In *Proceedings of the 38th International Conference on Software Engineering Companion.* 890–891.

[7] Jürgen Börstler, Mark S Hall, Marie Nordström, James H Paterson, Kate Sanders, Carsten Schulte, and Lynda Thomas. 2010. An evaluation of object oriented example programs in introductory programming textbooks. *ACM SIGCSE Bulletin* 41, 4 (2010), 126–143.

[8] Jürgen Börstler, Marie Nordström, and James H Paterson. 2011. On the quality of examples in introductory Java textbooks. *ACM Transactions on Computing Education* 11, 1 (2011), 3.

[9] Neil CC Brown and Amjad Altadmri. 2017. Novice Java programming mistakes: large-scale data vs. educator beliefs. *ACM Transactions on Computing Education* 17, 2 (2017).

[10] Simon Butler, Michel Wermelinger, Yijun Yu, and Helen Sharp. 2010. Exploring the influence of identifier names on code quality: An empirical study. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering.* 156–165.

[11] Norman Fenton and James Bieman. 2014. *Software metrics: a rigorous and practical approach.* CRC Press.

[12] Yossi Gil and Gal Lalouche. 2017. On the correlation between size and metric validity. *Empirical Software Engineering* 22, 5 (2017), 2585–2611.

[13] Robert Green and Henry Ledgard. 2011. Coding Guidelines: Finding the Art in the Science. *Commun. ACM* 54, 12 (2011), 57–63.

[14] Les Hatton. 2008. Testing the value of checklists in code inspections. *IEEE software* 25, 4 (2008), 82–88.

[15] Christopher D Hundhausen, Anukrati Agrawal, and Pawan Agarwal. 2013. Talking about code: Integrating pedagogical code reviews into early computing courses. *ACM Transactions on Computing Education* 13, 3 (2013), 14.

[16] Ronald Jabangwe, Jürgen Börstler, Darja Šmite, and Claes Wohlin. 2015. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. *Empirical Software Engineering* 20, 3 (2015), 640–693.

[17] Joint ACM/IEEE Task Force on Computing Curricula. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* Technical Report. ACM & IEEE Computer Society.

[18] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2017. Code quality Issues in Student Programs. In *Proceedings of the 22nd Annual Conference on Innovation and Technology in Computer Science Education.*

[19] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2016. Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 22rd Annual Conference on Innovation and Technology in Computer Science Education.* 41–46.

[20] Päivi Kinnunen and Beth Simon. 2010. Building theory about computing education phenomena: a discussion of grounded theory. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research.* ACM, 37–42.

[21] Sami Kollanus and Jussi Koskinen. 2009. Survey of Software Inspection Research. *The Open Software Engineering Journal* 3, 1 (2009), 15–34.

[22] Taek Lee and Jung-Been Lee. 2015. Effect analysis of coding convention violations on readability of post-delivered code. *IEICE TRANSACTIONS on Information and Systems* E98-D, 7 (2015), 1286–1296.

[23] T.C. Lethbridge, R.J. LeBlanc, A.E.K. Sobel, T.B. Hilburn, and J.L. Diaz-Herrera. 2006. SE2004: Recommendations for undergraduate software engineering curricula. *IEEE Software* 23, 6 (2006), 19–25.

[24] Romero Malaquias, Márcio Ribeiro, Rodrigo Bonifácio, Eduardo Monteiro, Flávio Medeiros, Alessandro Garcia, and Rohit Gheyi. 2017. The discipline of preprocessor-based annotations does# ifdef TAG n't# endif matter. In *Proceedings of the 25th International Conference on Program Comprehension.* 297–307.

[25] Robert C. Martin. 2008. *Clean Code: A Handbook of Agile Software Craftsmanship.* Prentice-Hall.

[26] José P Miguel, David Mauricio, and Glen Rodríguez. 2014. A review of software quality models for the evaluation of software products. *International Journal of Software Engineering & Applications* 5, 6 (2014), 31–53.

[27] Eric S. Raymond. 2010. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* www.snowballpublishing.com.

[28] Johannes Siedersleben. 2002. *Softwaretechnik: Praxiswissen fÃijr Softwareingenieure* (2 ed.). Carl Hanser Verlag.

[29] Saikrishna Sripada, Y Raghu Reddy, and Ashish Sureka. 2015. In support of peer code review and inspection in an undergraduate software engineering course. In *Software Engineering Education and Training (CSEET), 2015 IEEE 28th Conference on.* 3–6.

[30] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. 2014. Towards an empirically validated model for assessment of code quality. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research.* 99–108.

[31] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. 2016. Designing a rubric for feedback on code quality in programming courses. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research.* 160–164.

[32] Michael Striewe and Michael Goedicke. 2014. A review of static analysis approaches for programming exercises. In *International Computer Assisted Assessment Conference.* 100–113.

[33] Ted Tenny. 1988. Program readability: Procedures versus comments. *IEEE Transactions on Software Engineering* 14, 9 (1988), 1271–1279.

A INTERVIEW QUESTIONS

Table 3: List of interview questions

ID	Question text	Answer format
Q1	Gender?	M/F
Q2	In which country did you get most of your experience?	Short text
Q3-1	What is you current occupation and job title (if applicable)?	Student, professional programmer, educator
Q3-2	How many years of experience as a professional programmer do you have and how recent is that experience?	Number; to capture whether a student or educator could also be classified as developer
Q3-3	How many years of teaching software development do you have, in terms of full-time years?	Number; to capture whether a student or developer could also be classified as an educator
Q3-S1	What is your study program and level (e.g., Bachelor of Software Engineering)?	Short text
Q3-S2	Which study year are you in, in terms of full-time study equivalents?	Number
Q3-S3	Is programming one of your main study subjects?	Y/N
Q3-S4	How many programming courses did you take, i.e. courses with a significant programming component?	Number
Q3-S5	How many programming languages can you program in?	List of names
Q3-S6	How large was the largest program you developed?	Number
Q3-P1	What is your job title?	Short text
Q3-P2	Do your formal responsibilities involve quality assurance?	Y/N
Q3-P3	Which programming languages do you use most?	List of names
Q3-E1	How many courses related to programming do you teach per year on average?	Number
Q3-E2	Which courses related to programming did you teach during the last five years?	List of names
Q3-E3	Do you talk about code quality in those courses?	Y/N
Q3-E4	If yes, in which courses?	List of names
Q3-4	Which programming languages do you prefer most?	List of names
Q3-5	On a scale from strongly disagree to strongly agree, how much do you agree or disagree with the following statements regarding your personal experience related to software development?	
Q3-5a	I read and modify source code from other programmers.	7 item Likert-type scale
Q3-5b	Other people are reading and modifying the code that I write.	7 item Likert-type scale
Q3-5c	I review or comment other people's code.	7 item Likert-type scale
Q3-5d	Other people review or comment the code that I write.	7 item Likert-type scale
Q4	Please describe in detail, which properties or features you like or dislike with this code and how these properties or features affect the quality of the code. Please note that there are no correct or incorrect answers. We are primarily interested in code features that matter for you and why it does so.	Audio-recorded and transcribed
Q5	How would you define code quality? Which properties, features or indicator show you, personally, something about quality?	Text
Q6	On a scale from strongly disagree to strongly agree, how much do you agree or disagree with the following statements regarding your personal experience related to source code quality?	
Q6a	Code quality is of high importance in my work/studies/teaching.	7 item Likert-type scale
Q6b	I can easily tell good from bad code.	7 item Likert-type scale
Q6c	I regularly work with code quality issues.	7 item Likert-type scale
Q6d	I know how to measure code quality.	7 item Likert-type scale
Q6e	I have learned a lot about code quality during my education.	7 item Likert-type scale
Q6f	I have learned a lot about code quality from my colleagues.	7 item Likert-type scale
Q6g	I have learned a lot about code quality from the Internet.	7 item Likert-type scale

Table 4: List of interview questions (continued)

ID	Question text	Answer format
Q7	Please provide your top-3 recommendations for increasing the quality of code. Please indicate when a recommendation applies in special cases only.	
Q7-1	My top recommendation for achieving high code quality.	Short text
Q7-2	My second most important recommendation for achieving high code quality.	Short text
Q7-3	My third most important recommendation for achieving high code quality.	Short text
Q7-4	Any further important recommendations you want to mention?	Short text
Q8	According to your experience, what are the three topmost quality factors or indicators of high quality code?	
Q8-1	The most important quality factor/indicator for high quality code.	Short text
Q8-2	The second most important quality factor/indicator for high quality code.	Short text
Q8-3	The third most important quality factor/indicator for high quality code.	Short text
Q8-4	Any further highly important factors you want to mention?	Short text
Q9	According to your experience, what are the three most useful sources of information about code quality? Are these sources reliable and trustworthy?	
Q9-1	The most useful source of information about software quality.	Short text
Q9-2	The second most useful source of information about software quality.	Short text
Q9-3	The third most useful source of information about software quality.	Short text
Q9-4	Any further highly useful resources you want to mention?	Short text
Q10	According to your experience, what are the three most useful tools for improving code quality or achieving high quality code?	
Q10-1	The most useful tool for improving code quality.	Short text
Q10-2	The second most useful tool for improving code quality.	Short text
Q10-3	The third most useful tool for improving code quality.	Short text
Q10-4	Any further highly useful tools you want to mention?	Short text
Q11	Is there anything more you would like to bring up?	Text