A Systematic Mapping Study of Code Quality in Education – with Complete Bibliography

Hieke Keuning
Utrecht University
The Netherlands
h.w.keuning@uu.nl

Johan Jeuring
Utrecht University
The Netherlands
j.t.jeuring@uu.nl

Bastiaan Heeren
Open University of the Netherlands
The Netherlands
bastiaan.heeren@ou.nl

ABSTRACT

While functionality and correctness of code has traditionally been the main focus of computing educators, quality aspects of code are getting increasingly more attention. High-quality code contributes to the maintainability of software systems, and should therefore be a central aspect of computing education. We have conducted a systematic mapping study to give a broad overview of the research conducted in the field of code quality in an educational context. The study investigates paper characteristics, topics, research methods, and the targeted programming languages. We found 195 publications (1976–2022) on the topic in multiple databases, which we systematically coded to answer the research questions. This paper reports on the results and identifies developments, trends, and new opportunities for research in the field of code quality in computing education.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Software engineering education**.

KEYWORDS

programming education, software engineering education, code quality, refactoring, code smells, systematic mapping study

1 INTRODUCTION

Software quality is an important subject that Computer Science students need to learn during their studies. The quality of code, considering aspects such as naming, documentation, layout, control flow and structure, contributes to the readability, comprehensibility, and maintainability of software. Code style and quality is often discussed in the context of other software engineering topics, such as testing, code reviewing, and quality assurance (QA) in general. Automated assessment tools and tutoring systems might give feedback on code style, besides the correctness of solutions. Code quality has historically not been the main focus of educators [20, 102], possibly due to time, workload, lack of knowledge, and perceived lower importance. However, we have noticed an increase in interest in this topic, which we may (or may not) confirm with this study.

The goal of this Systematic Mapping Study [148] is to identify the landscape of studies that have been conducted on code quality in education. To our knowledge, this is the first overarching study on this topic. We first identify publication characteristics such as year, venue (journal/conference), followed by the topics, methods, languages, and relevant technical aspects. This paper makes the following contributions: (1) a large and complete list of papers on the topic, (2) a broad overview of the research area, and (3) an identification of research trends and new research opportunities.

This paper is an extension of a conference paper [100], and includes the complete reference list and coding.

2 BACKGROUND

In this section we give our definition of code quality for this study, and describe several relevant terms and aspects. We also briefly discuss other mapping studies related to computing education.

2.1 Terms and definitions

Software quality and *code quality* are sometimes intertwined, however, we consider code quality to be a more specific aspect of software quality. The ISO/IEC 25010 standard for software product quality comprises eight quality characteristics, among which functional suitability, usability, reliability, and maintainability. The last characteristic can be subdivided into modularity, reusability, analysability, modifiability, and testability. High-quality code can contribute to these characteristics.

Code quality is a term without a clear meaning and with various interpretations. We choose to focus on code quality as an aspect that appears *after* writing the initial program, dealing with analysing, reflecting on, and improving the program's static characteristics. We are interested in properties of source code that can be observed directly. As such, we focus on the *static* properties of code, as opposed to the *dynamic* properties such as correctness, test coverage, and runtime performance. We focus on the categories from the rubric designed by Stegeman et al. [177] to assess the quality of student code. These categories are documentation, layout, naming, flow, expressions, idiom, decomposition, and modularization.

Problems with these aspects are often denoted as *code smells*, a term introduced by Fowler [54]. Code smells may indicate a problem with the design of functionally correct code, affecting quality attributes of the software. Examples are duplication, dead code, overly complex code, and code with low cohesion and high coupling.

Refactoring is improving code step by step, while its functionality stays the same. Fowler's [54] well-known book describes a collection of refactorings, such as extracting a class or method, introducing an explaining variable, pull up a field or method, and replacing a magic number with a symbolic constant. *Design patterns* are reusable solutions to common problems in code [59], and can be used when refactoring.

To support developers with analysing and improving their code, many tools and systems are available. Tools such as PMD, Checkstyle, SonarQube, Resharper, and linters can automatically detect and report quality issues and code smells in a program. These tools often employ static analysis techniques to analyse code, although

static analysis has a broader application and can also be used to identify bugs and errors. There are also many tools to support the refactoring of code, often integrated in modern IDEs.

2.2 Related work

Systematic literature reviews, which dive deep into the literature on some topic, are increasingly being conducted for Computing Education (CEd) topics (see section 4.5 for examples related to our topic). A *systematic mapping study* aims to give a broad overview of a particular research area, usually by categorising its publications. While mapping studies are common in medicine, they are less common in other fields, such as software engineering [148] and CEd. A systematic mapping study on software testing in introductory programming courses by Scatalon et al. [168] is the most relevant to our study, because they also investigate a software quality aspect in an educational context. The authors selected 293 papers and categorised them on their topic and evaluation method.

Numerous mapping studies and literature reviews have been conducted on topics related to code quality, such as a mapping study on source code metrics [136], and a tertiary review on smells and refactoring [107]. However, these studies are not aimed at code quality in the context of *education*, the topic of our study.

3 METHOD

We generally follow the process from Petersen et al. [148] for doing systematic mapping studies in software engineering. We employ a different approach to classifying studies, as described in section 3.3.

3.1 Scope and research questions

The scope of this mapping study is:

> *Research on educational activities and support concerning code quality (as defined in 2.1), such as: instruction, analysis, assessment, tool support, tasks, and feedback.*

Within this scope we address the following research questions:

RQ1 Where are the papers published?
RQ2 Which topics have been addressed?
RQ3 Which types of studies have been conducted?
RQ4 For which programming language is the intervention?
RQ5 What are the trends over time?
RQ6 Which other topics are closely related to code quality?

3.2 Search process

The inclusion and exclusion criteria are defined in table 1. We have first assembled a base list of 40 papers that have been collected by ourselves over the years and meet the criteria. Two authors have verified that all publications on the list should be included.

3.2.1 Database search. We collected the keywords from the base papers, removing very general terms such as 'university' and 'examples', very specific terms such as names of tools and programming languages, and terms indicating the type of study. Based on these keywords we experimented with various search strings, checking whether the papers would end up in the search results. Because code quality is defined and named in various ways, we have used several specific terms in the search string to be as inclusive as possible.

During the process we have made the scope more clear by explicitly defining the edge topics for RQ6, as discussed in section 4.5.

We chose three databases, Scopus, ACM and IEEE, which cover a wide range of publications and allow searching with a complex search string. The search includes papers up to and including 2022. Because the final searches were conducted in December 2022, a few papers from 2022 could be missing. The final search string is shown below. We applied the search string to title, abstract, and keywords, and made some adjustments to match the database requirements. From our base list of 40 papers, 36 were found by this search.

```
    (programm* OR code OR coding OR software)
AND ("code quality" OR "software quality"
    OR "design quality" OR refactoring
    OR "static analysis" OR "software metrics"
    OR smell OR readability
    OR "code style" OR "coding style"
    OR "programming style")
AND (student OR teach* OR educat* OR curriculum OR novice)
```

Next, we elaborate on the process steps (summarised in figure 1).

Cleaning. One author combined the results from the three databases, and removed entries that are not papers, or are too short, and deleted duplicates based on title automatically using a script.

Pre-selection. One author filtered the list for exclusion based on title, and/or publication source, which were obviously out of scope because they are not about code quality and/or educational setting.

Selection. Two authors assessed a subset of the remaining list by reading title/abstract/keywords and selecting yes/no/maybe. Both 'yes' and 'maybe' indicated that we will consult the fulltext. If only one of the authors selected 'no', we discussed whether or not the fulltext should be consulted. We had three rounds of around 100 papers each, with an agreement of 77%, 78%, and 89%, respectively. One author assessed the remaining papers. For the fulltext selection we also checked and discussed several papers with two authors, after which one author selected the remaining papers. After this step, we had a selection of 168 papers for inclusion in our study.

3.2.2 Snowballing. The ambiguity surrounding the definition of code quality prevents constructing a search string that finds all relevant research. To find additional publications, we have performed *snowballing*: identifying relevant references from (backwards) and to (forwards) a set of papers [203]. For all 168 papers found in the previous steps, one author inspected all references from and to the paper (the latter using Google Scholar), and selected those within the scope. This inspection of thousands of references led to 27 additional papers. We stopped after one round of snowballing; we believe a second round would unlikely yield more relevant papers, because these papers would not have cited any of the papers from the database search. To answer RQ6, we kept a record of topics and papers referred to during snowballing that were outside our scope.

3.3 Coding

To answer RQs 2–4, we coded each paper in four categories: topic, aspect, method, and language. Codes are shown in a (box). We use the topics from the mapping study on testing in programming courses by Scatalon et al. [168] as our base for RQ2 (topic), with some small adjustments to fit our scope. We assigned one topic to each paper, representing its main focus or goal.
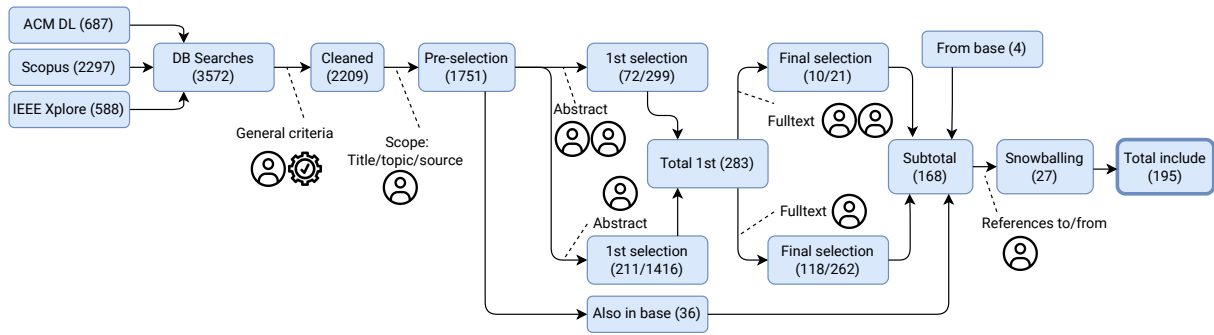
Figure 1: Paper selection process; person icon denotes one/two author(s); cog icon denotes an automated step.

Table 1: Inclusion and exclusion criteria.

	Include	Exclude
General	Scientific publications (journal papers and conference papers) in English.	Posters, papers shorter than four pages, theses, technical reports, books. Papers we cannot find. Papers preceding an extended version of a paper.
Topic	Publications that describe interventions in a formal educational context (high school/K-12, higher education).	Educational contexts aimed at professionals working in practice. Publications on automated feedback tools that provide style feedback alongside other error feedback, with no particular focus on code quality. Studies in which students are (among) the participants, but with no particular focus on education. Plagiarism. Static analysis tools used for assessing correctness etc.
Language	Code and design of general-purpose programming languages, teaching languages (e.g. Scratch).	Domain-specific languages such as SQL, low-level programming, very specific contexts (e.g. shader programming, or block-based robot programming).
Focus	A substantial part of the paper should be on code quality.	Interventions that only lead to improved code quality (among others), but are not specifically about code quality (further discussed in 4.5).
Quality	Publications should be indexed in Scopus, the ACM library, or IEEE library. Exceptions can be made for highly cited papers.	

- Curriculum The integration of code quality in the computing curriculum as a whole or in individual programming courses.
- Instruction:
  - Course materials
  - Programming assignments In addition, guidelines to conduct assignments related to code quality.
  - Programming process
  - Digital tools, either an external tool or selfmade tool.
  - Teaching methods Used when a paper addresses multiple of the instruction elements above.
- Learning outcome:
  - Program quality Assessment of students' submitted code.
  - Perceptions Students' (or teachers') attitudes towards code quality.
  - Behaviours Programming/refactoring behaviour. Broader than just program quality, but may include it.
  - Concept understanding Assessment of students' knowledge of code quality concepts.

For RQ2 we also identified for each paper whether it deals with one or more of the following domain-specific subtopics, attached when a topic is present in the title, abstract, or keywords of the paper: design patterns, refactoring, code/design smells, static analysis, readability, and metrics. These terms were taken from the keyword analysis, and the term 'readability' was most often used to refer to code quality by developers, educators, and students [20].

For the method (RQ3) we used categories from a recent Computing Education conference (Koli Calling 2021), shown below. A paper will only be coded by its main method.

Literature review Qualitative Case study
Descriptive/correlational Survey Quantative/other
System/Tool report Theory paper Experience report
(Quasi-)experimental Mixed methods Discussion paper

A subset of 11 papers were individually coded in all four categories by two authors, after which differences were discussed and resolved. One author coded the remaining papers.

4 RESULTS AND DISCUSSION

The full list and coding of the 195 papers can be found online in a searchable table[1] and in table 3. This section aggregates the findings and highlights examples from each category.

4.1 Paper characteristics (RQ1)

Figure 2 shows the publication years of the papers. The first publication appeared in 1976, but publications were rare in the 70s and 80s. After increasing only slightly in the 90s and 2000s, the attention for code quality in education clearly has been rising in the last decade.

Table 2 shows the main journals (33 papers) and conference proceedings (161 papers) in which the papers were published. The most

---

[1]www.hkeuning.nl/code-quality-mapping

Table 2: Publication venues (conference/journal) with at least 3 publications.

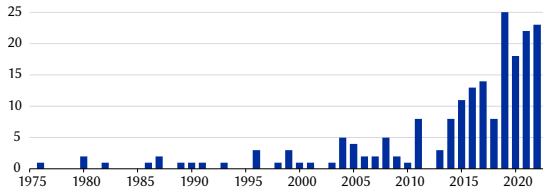| Name	C/J	Type	#
Special Interest Group on CS Ed. (SIGCSE)	C	Computing Education	21
Innovation and Technology in CS Ed. (ITiCSE)	C	Computing Education	11
Frontiers in Education (FIE)	C	Engineering Education	11
Koli Calling	C	Computing Education	7
Australasian Computing Education (ACE)	C	Computing Education	7
Computer-supported education (CSEDU)	C	Educational Technology	5
Computing Sciences in Colleges	J	Computing Education	4
Visual Languages and Human-Centric Computing	C	Human-Centric Comp.	4
IEEE Blocks & Beyond	C	Block programming	3
Systems and Software	J	Software Engineering	3
IEEE Access	J	General computing	3
Learning @ Scale	C	Educational Technology	3

Figure 2: Number of publications per year.

common venues are, as expected, related to Computing Education, however, the diversity of the remaining venues is broad. Papers on the topic have been published in venues on human-centric computing, software engineering, games, educational technology, program comprehension, and others.

4.2 Topics (RQ2) and methods (RQ3)

Figure 3 shows the correlation matrix of the topics and the methods. We did not find any literature reviews or theory papers, therefore these categories were omitted. We notice two major topics: program quality (41 papers) and tools (70 papers). We have made a distinction between tools created by the authors (59), and the use of an external tool (11). Figure 4 shows the correlation matrix of the topics and code quality aspects. All aspects clearly appear in multiple papers.

4.2.1 Curriculum. We have found only eight papers that revolve around integrating code quality into the (curriculum). As an example, Kirk et al. [102] study the prevalence of code quality in introductory programming courses. Techapalokul and Tilevich [181] advocate for the importance of integrating code quality in the teaching of programming in block-based environments, even though this code is usually not intended for practical use. Haendler and Neumann [71] present a framework for the assessment and training of refactoring.

4.2.2 Instruction. Overall, we observe that code quality in education revolves for a large part around digital (tools). Looking at the code quality aspects, we notice that those tools focus on several of them, such as identifying code smells and refactoring code, often using static analysis techniques. AUTOSTYLE [34] gives data-driven feedback on how to improve the style of correct programs step by step. Other recent tools are COMPARECFG [89], and a Java critiquer for antipatterns [192]. REFACTUTOR [73] is a tutoring system to

learn refactoring in Java. Keuning et al. [99] present a tutoring system in which students practice with improving functionally correct code, with the help of automated feedback and hints.

In some tools a 'gamification' approach was taken. Zsigmond et al. [208] present a system in which badges are awarded to students who adhere to good coding standards, using SONARQUBE for static analysis. Examples of badges are 'doc ace', 'complexity guru', and 'stylish coder'. PIRATE PLUNDER [161] is a game in which children learn to investigate and fix code smells in a Scratch environment.

We have also found papers on tools that focus on very specific aspects of code quality. For example, the EARTHWORM tool gives automated suggestions of decomposition of Python code [61]. FOOBAZ gives feedback on variable names [63]. Charitsis et al. [29] present a system based on machine learning techniques that can detect poor function names and suggest improvements.

Examples of external (professional) tools that are used in education are PMD [137] and CPPCHECK [44]. These tools can also be integrated in Continuous Integration practices, such as SONAR-QUBE [39]. We also noticed that selfmade tools often make use of external tools for specific tasks. For example, PYTA is a wrapper for PYLINT [110], adding custom code checks and improved error messages targeted at students. HYPERSTYLE [17] uses static analysers for different languages (PMD for Java, Detekt for Kotlin, and linters for JavaScript and Python), from which checks suitable for students are selected, categorized, and presented together with a grade.

Tools can be used to analyse large collections of student code (papers focussing on analysing program quality), and to support students in learning (papers focussing on a tool for instruction), and in some papers tools have a dual role: the authors conclude that student programs contain many flaws (identified by some tool), and therefore that tool could be used as an instructional aid [44]. However, it remains unclear whether these tools are suitable for educating novices, which is addressed by several papers. Nutbrown and Higgins [137] analyse differences between tool assessment and human assessment, and investigate the usefulness of such tools.

We have found only six papers that discuss (course materials). REFACTORY [70] is a non-digital card game to learn the principles of refactoring by resolving code smells. Other work analyses the readability of example programs in programming books [19].

Several papers discuss (programming assignments), for example, by presenting coding guidelines [206] and code readability best practices for students [164]. Stegeman et al. have developed a rubric for assessing the code quality of student code [177], which we described in more detail in section 2.1. Nandigam et al. [132] describe assignments in which students are instructed to explore and improve open source projects by measuring quality and applying refactorings where needed. Tempero and Tu [185] use code review assignments to asses how students understand the concept of 'maintainability'.

Nine papers discuss teaching about the (programming process). Stoecklin et al. [178] describe an approach for teaching refactoring through a series of incremental lessons. Abid et al. [2] present an experiment about the timing of refactoring in a student project. Lu et al. [115] introduce 'Continuous Inspection' of code quality in an
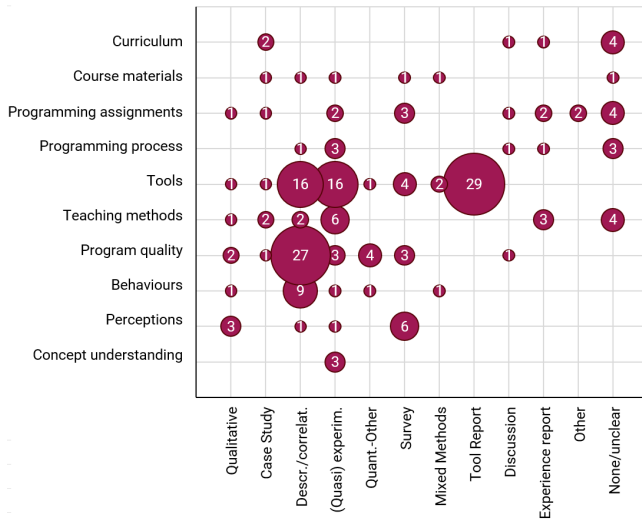
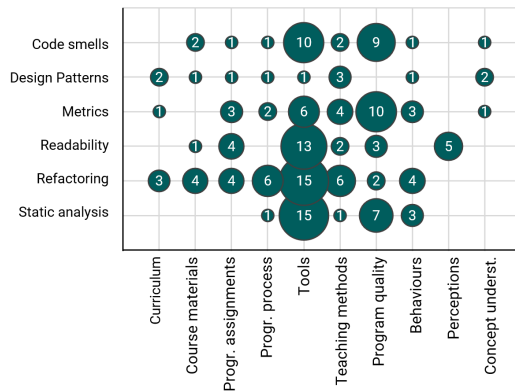**Figure 3: Correlation matrix of topics and methods.**

	Qualitative	Case Study	Descr./correlat.	(Quasi) experim.	Quant.-Other	Survey	Mixed Methods	Tool Report	Discussion	Experience report	Other	None/unclear
Curriculum			2						1	1		4
Course materials		1	1	1		1	1					1
Programming assignments	1	1	2		3				1	2	2	4
Programming process				3						1		3
Tools	1		16	16		4	2	29				
Teaching methods	1	2	2	6						3		4
Program quality	2		27	3	4	3			1			
Behaviours			9		1		1					
Perceptions	3			1	1	6						
Concept understanding				3								

Figure 4: Correlation matrix of topics and code quality aspects (a paper can have more than one aspect).

	Curriculum	Course materials	Progr. assignments	Progr. process	Tools	Teaching methods	Program quality	Behaviours	Perceptions	Concept underst.
Code smells		2	1	1	10	2	9	1		1
Design Patterns	2	1	1	1	1		3	1		2
Metrics	1		3	2	6	4	10	3		1
Readability		1		4	13	2	3		5	
Refactoring	3	4	4	6	15	6	2	4		
Static analysis				1	15	1	7	3		

educational setting. Passier et al. [145] describe how students can build an elegant JavaScript application step by step.

Papers labelled with (teaching method) address multiple instructional elements. Izu et al. [86] present a teaching resource, consisting of a textual explanation, a set of refactoring rules, and exercises, to help students with identifying code smells in conditional statements, and refactoring this code. Crespo et al. [37] focus on the concept of 'technical debt', and compare two different teaching methods: a penalisation (based on SonarQube metrics) and a rewarding strategy (with the metrics shown in a leaderbord).

4.2.3 Learning outcome. (Program quality) is a major category in papers, studying the programs that students write with respect to quality characteristics. These programs are mostly analysed automatically by a tool to identify code smells and calculate quality metrics. Examples of such large-scale studies, often analysing thousands of programs, are a study of PMD rule violations in Java programs [96], smells in Scratch programs [4], and indicators of

semantic style differences [43]. Cristea et al. [38] combine Formal Concept Analysis with Pylint, to detect issues with object-oriented design and too complex code. Groeneveld et al. [67] analyse the correlation between code quality and creativity, finding preliminary evidence for a larger number of issues in projects with high creativity. Grotov et al. [68] compare the coding style and complexity of Python programs written as regular scripts to code written in Jupyter computational notebooks. Ma and Tilevich [119] describe a set of anti-patterns that may arise when students move from Java to Python, but still write code in the much more verbose Java-style.

The majority of program quality studies employ quantitative, descriptive methods, but others take a qualitative approach. Some studies administer a survey to let teachers or students assess example code, for example to collect suggestions from expert programmers [83]. Andrade and Brunet [7] studied whether students were able to give useful feedback on the quality of other students' code.

A few papers study a specific phenomenon related to code quality, such as the 'unencapsulated collection' design smell [42]. Studies that assess student code quality by hand are more rare. Some papers compare code assessment by experts with code analysed by tools.

Thirteen papers focus on student (behaviours) with regards to code quality. Gilson et al. [62] observed how student Scrum teams deal with quality issues during a one-year project. Sripada and Reddy [175] also study student activities related to quality in multiple iterations of a development process. Senger et al. [171] replicate an earlier study with more and larger student programs, in which they run the static analyser FindBugs and study the correlation between the warnings found and correctness or struggling.

Eleven studies are on (perceptions) of teachers and students, of which five mention the term 'readability'. Kirk et al. [104] study high-school teachers' ideas and needs regarding code style and quality through interviews. Wiese et al. [199] investigate how beginner programmers assess the style of example programs, which they later replicate with a different student population [200]. Note that this is one of two replication studies we identified in our set of papers. An ITiCSE working group studied the differences in perceptions of code quality between developers, teachers, and students [20]. Fleury [53] conducted interviews with students asking them to evaluate and compare the style of several Java programs.

All three studies about (concept understanding) use a (quasi-) experimental approach. Hermans and Aivaloglou [77] study the effect of smells in Scratch code when students do comprehension tasks.

A few methods were not in our main list, such as 'educational design research' for iteratively designing a code quality rubric [177]. Recently, we observed the use of machine learning techniques [101].

4.3 Languages (RQ4)

Figure 5 shows a treemap with the languages that are targeted in the publications. Java and Python, popular general-purpose languages often used in teaching, are the most prevalent text-based languages in this study. Less expected might be the substantial number of papers dealing with programming in a block-based editor, such as Scratch [4, 161] and Snap! [88]. These papers investigate code smells or present learning tools. Other paradigms, such as functional programming, hardly appear.

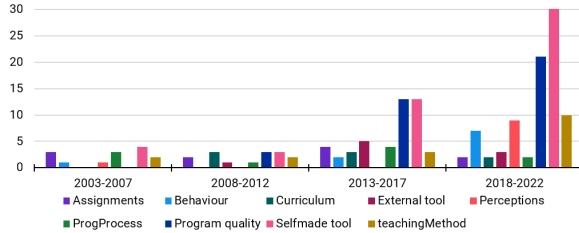**Figure 5: Treemap of targeted programming languages. Languages with less than 5 papers are omitted.**

Figure 6: Paper count per topic for the last 20 years, omitting topics with less than 8 papers.

4.4 Trends (RQ5)

Figure 6 shows the trends with respect to paper topics. We focus on the last twenty years, because we only have a small number of papers before that. We notice that much of the program quality research appeared in the last decade. The number of papers on tools has grown significantly, and the use of external tools is mostly a development from the last ten years. Studying perceptions is very much a recent development.

4.5 Related fields (RQ6)

As discussed before, the term *code quality* has no crystal clear definition. During our search for relevant publications, we regularly came across papers with a topic on the edges of our scope, as defined in section 3.1. In this section we list these topics, which are also relevant for learning and teaching about code quality, and refer to literature reviews on these topics, if available.

Software design education. While much of the research found in this study could be related to designing software, and in our definition we mention aspects such as decomposition and encapsulation, our mapping does not cover the broad field of teaching software design upfront. Instead, we focus on assessing the characteristics of the code after it has been written.

Design patterns education. We included some papers dealing with design patterns, because they were used as a means to refactor existing code. There are several other papers that focus on teaching and learning of design patterns in general.

Object-oriented programming. Abstraction, decomposition, and encapsulation are prominent topics in learning object-orientation, and contribute greatly to the quality of design and code.

Interventions leading to improved code quality. Some interventions may lead to improved code quality, but are not specifically about code quality. Examples are pair programming, test-driven development [168], and peer review [84].

Computational thinking. Abstraction, decomposition, and modularization are important aspects of computational thinking [80].

Code similarity and plagiarism. Rather the reverse of assessing the various ways a program can be written, several studies focus on code similarity, code clustering, and detecting plagiarism [135].

Program comprehension. This topic deals with the cognitive processes that programmers apply when trying to understand programs [169].

Automated assessment. Many systems for automated assessment of student programs incorporate some kind of style feedback [143].

4.6 Threats to validity

It is non-trivial to categorise a paper by its main method and topic. By only assigning one label, we might miss some additional topics and methods. Aspects were identified by looking for specific terms in the title and abstract; this simplified method might not correctly represent an article's main focus.

Although we performed an extensive database search followed by snowballing, we might have discarded relevant work based on an unclear title or abstract. Also, we have only included papers with code quality topics as their main focus. Because code quality can be integrated in software engineering courses, and is an element of overall software quality, we might miss some relevant research.

5 CONCLUSION

One of the earliest papers identified in this study on teaching programming style concludes with 'Perhaps the more recent structured languages such as PASCAL and C will make some of this emphasis less critical' [163]. Although tools and new languages simplify implementing good coding style, the author has not foreseen the ongoing issue with writing high-quality code.

We have conducted a systematic mapping study of code quality in education, which is the first overarching study on this topic. We identified and categorised 195 papers, studying paper characteristics, topics, domain-specific aspects, methods, and trends. Code quality is an upcoming topic with an increasing number of studies. Papers are published in a wide variety of venues on various topics. Its main focus has been on developing and evaluating tools for feedback on code smells, and suggestions for improvements and refactorings. Professional quality tools are increasingly being used in (and adapted for) education. Another major area is quality analysis of student code. We also observe that a growing number of studies target block-based programming environments, emphasising the need to start early with this topic. We have given several examples of the diversity in research, and shown related fields.

Because the goal of a mapping study is to give a broad overview, a possible direction for future work is to conduct a more in-depth literature study of a specific topic or aspect identified in this study. We would also encourage researchers to perform studies on the topics that have received little attention so far, such as integrating code quality into the computing curricula, developing and evaluating course materials, and studying student perceptions and behaviours.

REFERENCES

[1] Mohammad Abdallah and Mustafa Alrifaee. 2022. A Heuristic Tool for Measuring Software Quality Using Program Language Standards. *International Arab Journal of Information Technology* 19, 3 (2022), 314 – 322. DOI : https://doi.org/10.34028/iajit/19/3/4

[2] S. Abid, H.A. Basit, and N. Arshad. 2015. Reflections on teaching refactoring: A tale of two projects. *Proc. of ITiCSE* (2015). DOI : https://doi.org/10.1145/2729094.2742617

[3] Felix Adler, Gordon Fraser, Eva Grundinger, Nina Korber, Simon Labrenz, Jonas Lerchenberger, Stephan Lukasczyk, and Sebastian Schweikl. 2021. Improving Readability of Scratch Programs with Search-based Refactoring. *Proceedings - IEEE 21st International Working Conference on Source Code Analysis and Manipulation, SCAM 2021* (2021), 120 – 130. DOI : https://doi.org/10.1109/SCAM52516.2021.00023

[4] E. Aivaloglou and F. Hermans. 2016. How kids code and how we know: An exploratory study on the scratch repository. *Proc. of ICER* (2016). DOI : https://doi.org/10.1145/2960310.2960325

[5] K. Ala-Mutka, T. Uimonen, and H.-M. Jarvinen. 2004. Supporting students in C++ programming courses with automatic program style assessment. *Journal of Information Technology Education: Research* 3, 1 (2004). DOI : https://doi.org/10.28945/300

[6] Francisco Alfredo, André L Santos, and Nuno Garrido. 2022. Sprinter: A Didactic Linter for Structured Programming. In *Third International Computer Programming Education Conference (ICPEC 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[7] R. Andrade and J. Brunet. 2018. Can students help themselves? An investigation of students' feedback on the quality of the source code. *Proc. of FIE* (2018). DOI : https://doi.org/10.1109/fie.2018.8658503

[8] E. Araujo, D. Serey, and J. Figueiredo. 2016. Qualitative aspects of students' programs: Can we make them measurable? *Proc. of FIE* 2016-November (2016). DOI : https://doi.org/10.1109/FIE.2016.7757725

[9] Pasquale Ardimento, Mario Luca Bernardi, and Marta Cimitile. 2020. Software analytics to support students in object-oriented programming tasks: an empirical study. *IEEE Access* 8 (2020), 132171–132187. DOI : https://doi.org/10.1109/access.2020.3010172

[10] Daniel Avila, Edison Báez, Mireya Zapata, David López, Diego Zurita, and Danilo Martínez. 2021. Unreadable code in novice developers. In *World Conference on Information Systems and Technologies*. Springer, 46–51. DOI : https://doi.org/10.1007/978-3-030-72654-6_5

[11] Y. Bai, T. Wang, and H. Wang. 2019. Amelioration of Teaching Strategies by Exploring Code Quality and Submission Behavior. *IEEE Access* 7 (2019), 152744–152754. DOI : https://doi.org/10.1109/access.2019.2948640

[12] G. Balogh. 2015. Comparison of software quality in the work of children and professional developers based on their classroom exercises. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9159 (2015), 36–46. DOI : https://doi.org/10.1007/978-3-319-21413-9_3

[13] E. Baniassad, I. Beschastnikh, R. Holmes, G. Kiczales, and M. Allen. 2019. Learning to Listen for Design. In *Onward! 2019*. DOI : https://doi.org/10.1145/3359591.3359738

[14] D.J. Barnes and D. Shinners-Kennedy. 2011. A study of loop style and abstraction in pedagogic practice. *Conferences in Research and Practice in Information Technology Series* 114 (2011), 29–36.

[15] Tommaso Battistini, Nicolò Isaia, Andrea Sterbini, and Marco Temperini. 2022. DrPython-WEB: A Tool to Help Teaching Well-Written Python Programs. *Lecture Notes in Computer Science* 13230 LNCS (2022), 277 – 286. DOI : https://doi.org/10.1007/978-3-031-12429-7_20

[16] A.C. Benander and B.A. Benander. 1989. An empirical study of COBOL programs via a style analyzer: The benefits of good programming style. *The Journal of Systems and Software* 10, 4 (1989), 271–279. DOI : https://doi.org/10.1016/0164-1212(89)90074-5

[17] A. Birillo, I. Vlasov, A. Burylov, V. Selishchev, A. Goncharov, E. Tikhomirova, N. Vyahhi, and T. Bryksin. 2022. Hyperstyle: A Tool for Assessing the Code Quality of Solutions to Programming Assignments. *Proc. of SIGCSE* (2022), 307–313. DOI : https://doi.org/10.1145/3478431.3499294

[18] H. Blau and J.E.B. Moss. 2015. FrenchPress gives students automated feedback on Java program flaws. *Proc. of ITiCSE* 2015-June (2015), 15–20. DOI : https://doi.org/10.1145/2729094.2742622

[19] J. Börstler, M.E. Caspersen, and M. Nordström. 2016. Beauty and the Beast: on the readability of object-oriented example programs. *Software Quality Journal* 24, 2 (2016), 231–246.

[20] J. Börstler, H. Störrle, D. Toll, J. van Assema, R. Duran, S. Hooshangi, J. Jeuring, H. Keuning, C. Kleiner, and B. MacKellar. 2018. "I Know It When I See It" Perceptions of Code Quality. In *ITiCSE Conference Working Group Reports*. 70–85. DOI : https://doi.org/10.1145/3174781.3174785

[21] B.J. Bowman and W.A. Newman. 1990. Software metrics as a programming training tool. *The Journal of Systems and Software* 13, 2 (1990), 139–147. DOI : https://doi.org/10.1016/0164-1212(90)90119-7

[22] V. Bozhikova, M. Stoeva, B. Georgiev, and D. Nikolaeva. 2017. Improving the software quality - An educational approach. *International Scientific Conference Electronics, ET 2017 - Proceedings* 2017-January (2017), 1–4. DOI : https://doi.org/10.1109/ET.2017.8124337

[23] D.M. Breuker, J. Derriks, and J. Brunekreef. 2011. Measuring static quality of student code. *ITiCSE'11 - Proceedings of the 16th Annual Conference on Innovation and Technology in Computer Science* (2011), 13–17. DOI : https://doi.org/10.1145/1999747.1999754

[24] R.K. Brewer. 1976. Documentation Standards for Beginning Students. In *SIGCSE*. DOI : https://doi.org/10.1145/953026.803450

[25] Aline Brito, Andre Hora, and Marco Tulio Valente. 2022. Understanding Refactoring Tasks over Time: A Study Using Refactoring Graphs. *CIbSE 2022 - XXV Ibero-American Conference on Software Engineering* (2022). DOI : https://doi.org/10.5753/cibse.2022.20982

[26] B. Carlson. 2008. An Agile classroom experience: Teaching TDD and refactoring. *Proceedings - Agile 2008 Conference* (2008), 465–469. DOI : https://doi.org/10.1109/Agile.2008.39

[27] S. Celosmanovic and V. Ljubovic. 2022. JMetricGrader: A software for evaluating student projects using design object-oriented metrics and neural networks. *Jubilee International Convention on Information, Communication and Electronic Technology, MIPRO 2022 - Proceedings* (2022), 532 – 537. DOI : https://doi.org/10.23919/MIPRO55190.2022.9803776

[28] Laura Diana Cernau, Laura Silvia Dioșan, and Camelia Serban. 2022. A pedagogical approach in interleaving software quality concerns at an artificial intelligence course. *EASEAI 2022 - Proceedings of the 4th International Workshop on Education through Advanced Software Engineering and Artificial Intelligence, co-located with ESEC/FSE 2022* (2022), 18 – 24. DOI : https://doi.org/10.1145/3548660.3561332

[29] C. Charitsis, C. Piech, and J. C. Mitchell. 2022. Function Names: Quantifying the Relationship Between Identifiers and Their Functionality to Improve Them. In *Proc. of Learning @ Scale*. 93–101. DOI : https://doi.org/10.1145/3491140.3528269

[30] A. Chatzigeorgiou, N. Tsantalis, and I. Deligiannis. 2008. An empirical study on students' ability to comprehend design patterns. *Computers and Education* 51, 3 (2008), 1007–1016. DOI : https://doi.org/10.1016/j.compedu.2007.10.003

[31] H.-M. Chen, W.-H. Chen, and C.-C. Lee. 2018. An automated assessment system for analysis of coding convention violations in Java programming assignments*. *Journal of Information Science and Engineering* 34, 5 (2018), 1203–1221. DOI : https://doi.org/10.6688/JISE.201809_34(5).0006

[32] H. M. Chen, B. A. Nguyen, Y. X. Yan, and C. R. Dow. 2020. Analysis of Learning Behavior in an Automated Programming Assessment Environment: A Code Quality Perspective. *IEEE Access* 8 (2020), 167341–167354. DOI : https://doi.org/10.1109/ACCESS.2020.3024102

[33] Alexandru Chirvase, Laura Ruse, Mihnea Muraru, Mariana Mocanu, and Vlad Ciobanu. 2021. Clean Code - Delivering A Lightweight Course. *Proceedings - 2021 23rd International Conference on Control Systems and Computer Science Technologies, CSCS 2021* (2021), 420 – 423. DOI : https://doi.org/10.1109/CSCS52396.2021.00075

[34] R.R. Choudhury, H. Yin, and A. Fox. 2016. Scale-driven automatic hint generation for coding style. *Proc. of Intelligent Tutoring Systems* (2016). DOI : https://doi.org/10.1007/978-3-319-39583-8_12

[35] R.R. Choudhury, H. Yin, J. Moghadam, and A. Fox. 2016. AutoStyle: Toward coding style feedback at scale. *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW* 26-February-2016 (2016), 21–24. DOI : https://doi.org/10.1145/2818052.2874315

[36] Stanislav Chren, Martin Macák, Bruno Rossi, and Barbora Buhnova. 2022. Evaluating Code Improvements in Software Quality Course Projects. *EASE '22: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering* (2022), 160 – 169. DOI : https://doi.org/10.1145/3530019.3530036

[37] Y. Crespo, A. Gonzalez-Escribano, and M. Piattini. 2021. Carrot and Stick approaches revisited when managing Technical Debt in an educational context. In *Proc. of Technical Debt*. 99–108. DOI : https://doi.org/10.1109/techdebt52882.2021.00020

[38] D. Cristea, D. Șotropa, A. Molnar, and S. Motogna. 2021. On the Use of FCA Models in Static Analysis Tools to Detect Common Errors in Programming. *Lecture Notes in Computer Science* (2021), 3–18. DOI : https://doi.org/10.1007/978-3-030-86982-3_1

[39] P. H. de Andrade Gomes, R. E. Garcia, G. Spadon, D. M. Eler, C. Olivete, and R. C. Messias Correia. 2017. Teaching software quality via source code inspection tool. In *Proc. of FIE*. DOI : https://doi.org/10.1109/fie.2017.8190658

[40] D.M.N. de Araújo, D.M. Eler, and R.E. Garcia. 2020. Teacher Mate: A Support Tool for Teaching Code Quality. *Advances in Intelligent Systems and Computing* 1134 (2020), 407–413. DOI : https://doi.org/10.1007/978-3-030-43020-7_54

[41] E.S.J. De Faria, J.M. Adán-Coello, and K. Yamanaka. 2006. Forming groups for collaborative learning in introductory computer programming courses based on students' programming styles: An empirical study. *Proc. of FIE* (2006). DOI : https://doi.org/10.1109/FIE.2006.322313

[42] G. De Ruvo, E. Tempero, A. Luxton-Reilly, and N. Giacaman. 2018. Unencapsulated collection - A teachable design smell. *Proc. of SIGCSE* (2018). DOI : https://doi.org/

10.1145/3159450.3159469

[43] G. De Ruvo, E. Tempero, G.B. Rowe, and N. Giacaman. 2018. Understanding Semantic Style by Analysing Student Code. *Proc. of ACE* (2018). DOI : https://doi.org/10.1145/3160489.3160500

[44] T. Delev and D. Gjorgjevikj. 2017. Static analysis of source code written by novice programmers. *Proc. of EDUCON* (2017). DOI : https://doi.org/10.1109/educon.2017.7942942

[45] S. Demeyer, F. Van Rysselberghe, T. Gîrba, J. Ratzinger, R. Marinescu, T. Mens, B. Du Bois, D. Janssens, S. Ducasse, M. Lanza, M. Rieger, H. Gall, and M. El-Ramly. 2005. The LAN-simulation: A refactoring teaching example. *International Workshop on Principles of Software Evolution (IWPSE)* 2005 (2005), 123–131. DOI : https://doi.org/10.1109/IWPSE.2005.30

[46] Steffen Dick, Stefan Schulz, and Christoph Bockisch. 2022. A study on the quality mindedness of students. *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)* P-321 (2022), 119 – 124. DOI : https://doi.org/10.18420/SEUH2022_12

[47] H.M. Dos Santos, E. Figueiredo, V.H.S. Durelli, L.T. Da Silva, M. Souza, and R.S. Durelli. 2019. Cleangame: Gamifying the identification of code smells. *SBES '19: Proceedings of the XXXIII Brazilian Symposium on Software Engineering* (2019), 437–446. DOI : https://doi.org/10.1145/3350768.3352490

[48] S.H. Edwards, N. Kandru, and M.B.M. Rajagopal. 2017. Investigating static analysis errors in student Java programs. *ICER 2017 - Proceedings of the 2017 ACM Conference on International Computing Education Research* (2017), 65–73. DOI : https://doi.org/10.1145/3105726.3106182

[49] T. Effenberger and R. Pelánek. 2022. Code Quality Defects Across Introductory Programming Topics. *SIGCSE* (2022), 941 – 947. DOI : https://doi.org/10.1145/3478431.3499415

[50] Davide Falessi and Philippe Kruchten. 2015. Five reasons for including technical debt in the software engineering curriculum. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*. 1–4. DOI : https://doi.org/10.1145/2797433.2797462

[51] A. Fehnker and R. de Man. 2019. Detecting and Addressing Design Smells in Novice Processing Programs. *Comm. in Computer and Information Science* (2019). DOI : https://doi.org/10.1007/978-3-030-21151-6_24

[52] Gerhard Fischer. 1987. *A critic for LISP.* Technical Report. COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE. DOI : https://doi.org/10.21236/ada446617

[53] A. Fleury. 2001. Encapsulation and reuse as viewed by java students. *Proc. of SIGCSE* (2001). DOI : https://doi.org/10.1145/366413.364582

[54] M. Fowler. 1999. *Refactoring: improving the design of existing code.* Addison-Wesley Professional. DOI : https://doi.org/10.1007/3-540-45672-4_31

[55] Gordon Fraser, Ute Heuer, Nina Körber, Florian Obermüller, and Ewald Wasmeier. 2021. Litterbox: A linter for scratch programs. In *IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 183–188. DOI : https://doi.org/10.1109/icse-seet52601.2021.00028

[56] N. Funabiki, T. Ogawa, N. Ishihara, M. Kuribayashi, and W. Kao. 2016. A Proposal of Coding Rule Learning Function in Java Programming Learning Assistant System. In *International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*. 561–566. DOI : https://doi.org/10.1109/CISIS.2016.94

[57] Iris Gaber and Amir Kirsh. 2018. The Effect of Reporting Known Issues on Students' Work. In *SIGCSE*. DOI : https://doi.org/10.1145/3159450.3159456

[58] Iris Gaber and Amir Kirsh. 2021. Using examples as guideposts for programming exercises: Providing support while preserving the challenge. *ICCSE 2021 - IEEE 16th International Conference on Computer Science and Education* (2021), 391 – 397. DOI : https://doi.org/10.1109/ICCSE51940.2021.9569541

[59] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional.

[60] H. Gardner. 2004. Design patterns in scientific software. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3045 (2004), 776–785. DOI : https://doi.org/10.1007/978-3-540-24767-8_82

[61] N. Garg and A.W. Keen. 2018. Earthworm: Automated Decomposition Suggestions. In *Proc. of Koli Calling*. DOI : https://doi.org/10.1145/3279720.3279736

[62] F. Gilson, M. Morales-Trujillo, and M. Mathews. 2020. How junior developers deal with their technical debt? *Proc. of Technical Debt* (2020). DOI : https://doi.org/10.1145/3387906.3388624

[63] E.L. Glassman, L. Fischer, J. Scott, and R.C. Miller. 2015. Foobaz: Variable Name Feedback for Student Code at Scale. In *Proc. of User Interface Software & Techn.* DOI : https://doi.org/10.1145/2807442.2807495

[64] Pedro Henrique Gomes, Rogerio Eduardo Garcia, Danilo Medeiros Eler, Ronaldo Celso Correia, and Celso Olivete Junior. 2021. Software Quality as a Subsidy for Teaching Programming. *Proc. of FIE* 2021-October (2021). DOI : https://doi.org/10.1109/FIE49875.2021.9637475

[65] M.A. Gómez-Martín, G. Jiménez-Díaz, and J. Arroyo. 2009. Teaching design patterns using a family of games. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE* (2009), 268–272. DOI : https://doi.org/10.1145/1562877.1562960

[66] G. Grigas. 1995. Investigation of the relationship between program correctness and programming style. *Informatica (Netherlands)* 6, 3 (1995), 265–276. DOI : https://doi.org/10.3233/INF-1995-6302

[67] W. Groeneveld, D. Martin, T. Poncelet, and K. Aerts. 2022. Are Undergraduate Creative Coders Clean Coders? A Correlation Study. *SIGCSE* (2022), 314–320. DOI : https://doi.org/10.1145/3478431.3499345

[68] K. Grotov, S. Titov, V. Sotnikov, Y. Golubev, and T. Bryksin. 2022. A Large-Scale Comparison of Python Code in Jupyter Notebooks and Scripts. *Proc. of Mining Software Repositories Conference (MSR)* (2022), 353–364. DOI : https://doi.org/10.1145/3524842.3528447

[69] H. Gu and S. K. Dubey. 2014. Academic coding guideline model - OCG. In *Conference on Computing for Sustainable Global Development (INDIACom)*. DOI : https://doi.org/10.1109/indiacom.2014.6828032

[70] T. Haendler. 2019. A card game for learning software-refactoring principles. *Proc. of GamiLearn* (2019).

[71] T. Haendler and G. Neumann. 2019. A framework for the assessment and training of software refactoring competences. In *Proc. of KMIS*. DOI : https://doi.org/10.5220/0008350803070316

[72] Thorsten Haendler and Gustaf Neumann. 2019. Serious refactoring games. In *Proceedings of the 52nd Hawaii International Conference on System Sciences.* DOI : https://doi.org/10.24251/hicss.2019.927

[73] T. Haendler, G. Neumann, and F. Smirnov. 2020. RefacTutor: An Interactive Tutoring System for Software Refactoring. *Communications in Computer and Information Science* (2020). DOI : https://doi.org/10.1007/978-3-030-58459-7_12

[74] Sivana Hamer, Christian Quesada-López, and Marcelo Jenkins. 2021. Students Projects' Source Code Changes Impact on Software Quality Through Static Analysis. *Communications in Computer and Information Science* 1439 CCIS (2021), 553 – 564. DOI : https://doi.org/10.1007/978-3-030-85347-1_39

[75] Sivana Hamer, Christian Quesada-López, Alexandra Martínez, and Marcelo Jenkins. 2021. Measuring students' source code quality in software development projects through commit-impact analysis. In *International Conference on Information Technology & Systems*. Springer, 100–109. DOI : https://doi.org/10.1007/978-3-030-68418-1_11

[76] H. Hashiura, S. Matsuura, and S. Komiya. 2010. A tool for diagnosing the quality of java program and a method for its effective utilization in education. *Proceedings of the 9th WSEAS International Conference on Applications of Computer Engineering, ACE '10* (2010), 276–282. https://www.scopus.com/inward/record.uri?eid=2-s2.0-79952590241&partnerID=40&md5=9c56b08bc904d200bed0bf2ee2465b3f

[77] F. Hermans and E. Aivaloglou. 2016. Do code smells hamper novice programming? A controlled experiment on Scratch programs. *Proc. of ICPC* (2016). DOI : https://doi.org/10.1109/icpc.2016.7503706

[78] F. Hermans, K.T. Stolee, and D. Hoepelman. 2016. Smells in block-based programming languages. *VL/HCC* (2016). DOI : https://doi.org/10.1109/vlhcc.2016.7739666

[79] S. Hooshangi and S. Dasgupta. 2017. Code quality: Examining the efficacy of automated tools. *AMCIS 2017 - America's Conference on Information Systems: A Tradition of Innovation* 2017-August (2017). https://www.scopus.com/inward/record.uri?eid=2-s2.0-85048410727&partnerID=40&md5=38e1128a67cb6f7dccb9926ddc5b0d57

[80] T. Hsu, S. Chang, and Y. Hung. 2018. How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education* 126 (2018), 296–310. DOI : https://doi.org/10.1016/j.compedu.2018.07.004

[81] Z. Hu and E.F. Gehringer. 2019. Improving Feedback on GitHub Pull Requests: A Bots Approach. *Proc. of FIE* 2019-October (2019). DOI : https://doi.org/10.1109/FIE43999.2019.9028685

[82] S.-L. Hung, I.-F. Kwok, and R. Chan. 1993. Automatic programming assessment. *Computers and Education* 20, 2 (1993), 183–190. DOI : https://doi.org/10.1016/0360-1315(93)90086-X

[83] M. Ichinco, A. Zemach, and C. Kelleher. 2013. Towards generalizing expert programmers' suggestions for novice programmers. *VL/HCC* (2013). DOI : https://doi.org/10.1109/vlhcc.2013.6645259

[84] T. Indriasari, A. Luxton-Reilly, and P. Denny. 2020. A review of peer code review in higher education. *ACM Trans. on Computing Education (TOCE)* (2020), 1–25. DOI : https://doi.org/10.1145/3403935

[85] Y. Ito, A. Hazeyama, Y. Morimoto, H. Kaminaga, S. Nakamura, and Y. Miyadera. 2014. A Method for Detecting Bad Smells and ITS Application to Software Engineering Education. In *Proc. of International Conference on Advanced Applied Informatics (IIAI)*. 670–675. DOI : https://doi.org/10.1109/IIAI-AAI.2014.139

[86] C. Izu, P. Denny, and S. Roy. 2022. A Resource to Support Novices Refactoring Conditional Statements. *Proc. of ITiCSE* (2022), 344–350. DOI : https://doi.org/10.1145/3502718.3524810

[87] J. Jansen, A. Oprescu, and M. Bruntink. 2017. The impact of automated code quality feedback in programming education. *SATToSE* (2017).

[88] S. Jatzlau, S. Seegerer, and R. Romeike. 2019. The Five Million Piece Puzzle: Finding Answers in 500,000 Snap!-Projects. In *IEEE Blocks and Beyond Workshop.* DOI : https://doi.org/10.1109/bb48857.2019.8941206

[89] L. Jiang, R. Rewcastle, P. Denny, and E. Tempero. 2020. CompareCFG: Providing Visual Feedback on Code Quality Using Control Flow Graphs. In *ITiCSE.* DOI : https://doi.org/10.1145/3341525.3387362

[90] Saj-Nicole A Joni and Elliot Soloway. 1986. But my program runs! Discourse rules for novice programmers. *Journal of Educational Computing Research* 2, 1 (1986), 95–125. DOI : https://doi.org/10.2190/6e5w-ar7c-nx76-hut2

[91] Oscar Karnalim and Simon. 2021. Promoting Code Quality via Automated Feedback on Student Submissions. *Proc. of FIE* 2021-October (2021). DOI : https://doi.org/10.1109/FIE49875.2021.9637193

[92] Sai Anirudh Karre, Lalit Mohan Sanagavarapu, and Y. Raghu Reddy. 2021. Using project-based approach to teach design patterns: An Experience Report. *ISEC 2021: 14th Innovations in Software Engineering Conference (formerly known as India Software Engineering Conference)* (2021). DOI : https://doi.org/10.1145/3452383.3452399

[93] R. Kasahara, K. Sakamoto, H. Washizaki, and Y. Fukazawa. 2019. Applying gamification to motivate students to write high-quality code in programming assignments. *Proc. of ITiCSE* (2019), 92–98. DOI : https://doi.org/10.1145/3304221.3319792

[94] M. Katić, I. Botički, and K. Fertalj. 2013. Impact of aspect-oriented programming on the quality of novices' programs: A comparative study. *Journal of Information and Organizational Sciences* 37, 1 (2013), 45–61. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84879267673&partnerID=40&md5=1bf982f83582bb2be46e3d6b567edf7e

[95] A. Keen and K. Mammen. 2015. Program decomposition and complexity in CS1. *SIGCSE* (2015), 48–53. DOI : https://doi.org/10.1145/2676723.2677219

[96] H. Keuning, B. Heeren, and J. Jeuring. 2017. Code quality issues in student programs. *Proc. of ITiCSE* (2017). DOI : https://doi.org/10.1145/3059009.3059061

[97] H. Keuning, B. Heeren, and J. Jeuring. 2019. How teachers would help students to improve their code. *ITiCSE* (2019). DOI : https://doi.org/10.1145/3304221.3319780

[98] H. Keuning, B. Heeren, and J. Jeuring. 2020. Student Refactoring Behaviour in a Programming Tutor. *Proc. of Koli Calling* (2020). DOI : https://doi.org/10.1145/3428029.3428043

[99] H. Keuning, B. Heeren, and J. Jeuring. 2021. A Tutoring System to Learn Code Refactoring. *Proc. of SIGCSE* (2021), 562–568. DOI : https://doi.org/10.1145/3408877.3432526

[100] H. Keuning, J. Jeuring, and B. Heeren. 2023. A Systematic Mapping Study of Code Quality in Education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. DOI : https://doi.org/10.1145/3587102.3588777

[101] W. Kim, S. Rhim, J. Choi, and K. Han. 2020. Modeling Learners' Programming Skills and Question Levels Through Machine Learning. In *Proc. of HCI*. 281–288. DOI : https://doi.org/10.1007/978-3-030-60703-6_36

[102] D. Kirk, T. Crow, A. Luxton-Reilly, and E. Tempero. 2020. On assuring learning about code quality. *Proc. of ACE* (2020). DOI : https://doi.org/10.1145/3373165.3373175

[103] Diana Kirk, Tyne Crow, Andrew Luxton-Reilly, and Ewan Tempero. 2021. Mind the Gap: Searching for Clarity in NCEA. *Proc. of ITiCSE* (2021), 192 – 198. DOI : https://doi.org/10.1145/3430665.3456318

[104] D. Kirk, T. Crow, A. Luxton-Reilly, and E. Tempero. 2022. Teaching code quality in high school programming courses - Understanding teachers' needs. *Proc. of ACE* (2022), 36–45. DOI : https://doi.org/10.1145/3511861.3511866

[105] D. Kirk, E. Tempero, A. Luxton-Reilly, and T. Crow. 2020. High School Teachers' Understanding of Code Style. *Koli Calling* (2020). DOI : https://doi.org/10.1145/3428029.3428047

[106] F. Koetter, M. Kochanowski, M. Kintz, B. Kersjes, I. Bogicevic, and S. Wagner. 2019. Assessing software quality of agile student projects by data-mining software repositories. *CSEDU 2019 - Proceedings of the 11th International Conference on Computer Supported Education* 2 (2019), 244–251. DOI : https://doi.org/10.5220/0007688602440251

[107] G. Lacerda, F. Petrillo, M. Pimenta, and Y. Guéhéneuc. 2020. Code smells and refactoring: A tertiary systematic review of challenges and observations. *J. of Systems and Software* (2020). DOI : https://doi.org/10.1016/j.jss.2020.110610

[108] J.W. Lartigue and R. Chapman. 2018. Comprehension and application of design patterns by novice software engineers. *Proceedings of the ACMSE 2018 Conference* 2018-January (2018). DOI : https://doi.org/10.1145/3190645.3190686

[109] Xiaosong Li and Christine Prasad. 2005. Effectively teaching coding standards in programming. In *Proceedings of the 6th conference on Information technology education*. 239–244. DOI : https://doi.org/10.1145/1095714.1095770

[110] D. Liu and A. Petersen. 2019. Static analyses in python programming courses. *Proc. of SIGCSE* (2019). DOI : https://doi.org/10.1145/3287324.3287503

[111] X. Liu and G. Woo. 2020. Applying Code Quality Detection in Online Programming Judge. *Proceedings of the 2020 5th International Conference on Intelligent Information Technology* (2020), 56–60. DOI : https://doi.org/10.1145/3385209.3385226

[112] Yi Liu, Jenq-Foung Yao, Gita Williams, and Gerald Adkins. 2007. Studying Software Metrics Based on Real-World Software Systems. *J. Comput. Sci. Coll.* 22, 5 (2007), 55–61.

[113] C. Lopez, J.M. Alonso, R. Marticorena, and J.M. Maudes. 2014. Design of e-activities for the learning of code refactoring tasks. *International Symposium on Computers in Education, SIIE* (2014), 35–40. DOI : https://doi.org/10.1109/siie.2014.7017701

[114] Shaoxiao Lu, Xu Wang, Haici Zhou, Qing Sun, Wenge Rong, and Ji Wu. 2021. Anomaly Detection for Early Warning in Object-oriented Programming Course. *TALE 2021 - IEEE International Conference on Engineering, Technology and Education, Proceedings* (2021), 204 – 211. DOI : https://doi.org/10.1109/TALE52509.2021.9678677

[115] Y. Lu, X. Mao, T. Wang, G. Yin, and Z. Li. 2019. Improving students' programming quality with the continuous inspection process: a social coding perspective. *Frontiers of Computer Science* 14, 5 (2019). DOI : https://doi.org/10.1007/s11704-019-9023-2

[116] Nikola Luburić, Dragan Vidaković, Jelena Slivka, Simona Prokić, Katarina-Glorija Grujić, Aleksandar Kovačević, and Goran Sladić. 2022. Clean Code Tutoring: Makings of a Foundation. *International Conference on Computer Supported Education, CSEDU - Proceedings* 1 (2022), 137 – 148. DOI : https://doi.org/10.5220/0010800900003182

[117] Nikola Luburic, Balša Šarenac, Luka Doric, Dragan Vidakovic, Katarina-Glorija Grujic, Aleksandar Kovacevic, and Simona Prokic. 2022. Challenges of knowledge component modeling: A software engineering case study. *International Conference on Higher Education Advances* 2022-June (2022), 901 – 908. DOI : https://doi.org/10.4995/HEAd22.2022.14217

[118] Roope Luukkainen, Jussi Kasurinen, Uolevi Nikula, and Valentina Lenarduzzi. 2022. ASPA: A Static Analyser to Support Learning and Continuous Feedback on Programming Courses. An Empirical Validation. *Proceedings - International Conference on Software Engineering* (2022), 29 – 39. DOI : https://doi.org/10.1109/ICSE-SEET55299.2022.9794188

[119] Y. Ma and E. Tilevich. 2021. You have said too much : Java-like verbosity anti-patterns in python codebases. *Proc. of the SPLASH-E Symposium* (2021), 13–18. DOI : https://doi.org/10.1145/3484272.3484960

[120] Terumasa Maeta and Shimpei Matsumoto. 2022. An Applicability Study on Refactoring Principles in Reading-Based Programming Learning. *Proceedings - 2022 12th International Congress on Advanced Applied Informatics, IIAI-AAI 2022* (2022), 264 – 267. DOI : https://doi.org/10.1109/IIAIAAI55812.2022.00060

[121] Sami Mäkelä and Ville Leppänen. 2004. Japroch: A tool for checking programming style. *Koli Calling* (2004), 151.

[122] Andrew D. Marshall, Michael J. Katchabaw, and Michael A. Bauer. 1996. Using software metrics tools for maintenance decisions: a classroom exercise. *Proceedings of the International Symposium on Assessment of Software Tools* (1996), 47–58. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0029703728&partnerID=40&md5=aac51a49622a5174db9a6af0e9dd2302

[123] K. McMaster, S. Sambasivam, and S. Wolthuis. 2013. Teaching programming style with ugly code. *ISECON* (2013).

[124] S.A. Mengel and V. Yerramilli. 1999. A case study of the static analysis of the quality of novice student programs. *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)* 31, 1 (1999), 78–82. DOI : https://doi.org/10.1145/384266.299689

[125] Susan A. Mengel and Joseph Ulans. 1998. Using Verilog LOGIS-COPE to analyze student programs. *Proc. of FIE* 3 (1998), 1213–1218. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0032308219&partnerID=40&md5=b6bad2060c27c827312708ece3541ab5

[126] S. A. Mengel and J. V. Ulans. 1999. A case study of the analysis of novice student programs. In *Proc. of Conference on Software Engineering Education and Training*. 40–49. DOI : https://doi.org/10.1109/CSEE.1999.755178

[127] Greg Michaelson. 1996. Automatic analysis of functional program style. In *Software Engineering Conference, Australian*. IEEE Computer Society, 38–38.

[128] K. Mierle, S. Roweis, and G. Wilson. 2005. Mining student CVS repositories for performance indicators. *Proceedings of the 2005 International Workshop on Mining Software Repositories, MSR 2005* (2005). DOI : https://doi.org/10.1145/1083142.1083150

[129] A.-J. Molnar, S. Motogna, and C. Vlad. 2020. Using static analysis tools to assist student project evaluation. *EASEAI 2020 - Proceedings of the 2nd ACM SIGSOFT International Workshop on Education through Advanced Software Engineering and Artificial Intelligence, Co-located with ESEC/FSE 2020* (2020), 7–12. DOI : https://doi.org/10.1145/3412453.3423195

[130] Jesús Moreno and Gregorio Robles. 2014. Automatic detection of bad programming habits in scratch: A preliminary study. In *Proc. of FIE*. IEEE, 1–4. DOI : https://doi.org/10.1109/fie.2014.7044055

[131] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. 2015. Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia* 46 (2015), 1–23.

[132] J. Nandigam, V. N. Gudivada, and A. Hamou-Lhadj. 2008. Learning software engineering principles using open source software. In *Proc. of FiE*. DOI : https://doi.org/10.1109/fie.2008.4720643

[133] M. Nascimento, E. Araújo, D. Serey, and J. Figueiredo. 2020. The Role of Source Code Vocabulary in Programming Teaching and Learning. In *Proc. of FIE*. 1–8. DOI : https://doi.org/10.1109/fie44824.2020.9274137

[134] Huy Nguyen, Michelle Lim, Steven Moore, Eric Nyberg, Majd Sakr, and John Stamper. 2021. Exploring Metrics for the Analysis of Code Submissions in an Introductory Data Science Course. In *LAK21: 11th International Learning Analytics and Knowledge Conference*. 632–638. DOI : https://doi.org/10.1145/3448139.3448209

[135] M. Novak, M. Joy, and D. Kermek. 2019. Source-code similarity detection and detection tools used in academia: a systematic review. *ACM Transactions on Computing Education (TOCE)* 19, 3 (2019), 1–37. DOI : https://doi.org/10.1145/3313290

[136] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo. 2017. Source code metrics: A systematic mapping study. *J. Syst. and Softw.* (2017). DOI : https://doi.org/10.1016/j.jss.2017.03.044

[137] S. Nutbrown and C. Higgins. 2016. Static analysis of programming exercises: Fairness, usefulness and a method for application. *Computer Science Ed.* (2016). DOI : https://doi.org/10.1080/08993408.2016.1179865

[138] Florian Obermüller, Lena Bloch, Luisa Greifenstein, Ute Heuer, and Gordon Fraser. 2021. Code Perfumes: Reporting Good Code to Encourage Learners. *WiPSCE '21: The 16th Workshop in Primary and Secondary Computing Education* (2021). DOI : https://doi.org/10.1145/3481312.3481346

[139] M. Ohtsuki and T. Kakeshita. 2019. Utilizing software engineering education support system ALECSS at an actual software development experiment: A case study. *CSEDU 2019 - Proceedings of the 11th International Conference on Computer Supported Education* 2 (2019), 367–375. DOI : https://doi.org/10.5220/0007723203670375

[140] M. Ohtsuki, K. Ohta, and T. Kakeshita. 2016. Software engineer education support system ALECSS utilizing devOps tools. *iiWAS '16: Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services* (2016), 209–213. DOI : https://doi.org/10.1145/3011141.3011200

[141] P.W. Oman and C.R. Cook. 1991. A programming style taxonomy. *The Journal of Systems and Software* 15, 3 (1991). DOI : https://doi.org/10.1016/0164-1212(91)90044-7

[142] J. Walker Orr. 2022. Automatic Assessment of the Design Quality of Student Python and Java Programs. *J. Comput. Sci. Coll.* 38, 1 (dec 2022), 27–36.

[143] J. C. Paiva, J. P. Leal, and Á. Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. on Computing Education (TOCE)* (2022), 1–40. DOI : https://doi.org/10.1145/3513140

[144] B. Park, H. Tak, and H. G. Cho. 2014. Structural Analysis of Source Code Collected from Programming Contests. In *Conference on Computer and Information Technology*. DOI : https://doi.org/10.1109/cit.2014.171

[145] H. Passier, S. Stuurman, and H. Pootjes. 2014. Beautiful JavaScript: How to Guide Students to Create Good and Elegant Code. In *Proc. of CSERC*. DOI : https://doi.org/10.1145/2691352.2691358

[146] Evan W Patton, Audrey Seo, and Franklyn Turbak. 2019. Enhancing Abstraction in App Inventor with Generic Event Handlers. In *IEEE Blocks and Beyond Workshop (B&B)*. IEEE, 63–71. DOI : https://doi.org/10.1109/bb48857.2019.8941193

[147] J. Perretta, W. Weimer, and A. DeOrio. 2019. Human vs. Automated coding style grading in computing education. *ASEE Annual Conference and Exposition, Conference Proceedings* (2019). https://www.scopus.com/inward/record.uri?eid=2-s2.0-85078787387&partnerID=40&md5=3b3eb8ad58b650fb6e0546d89c60fd2f

[148] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. 2008. Systematic mapping studies in software engineering. In *Proc. of Evaluation and Assessment in SE*. DOI : https://doi.org/10.14236/ewic/ease2008.8

[149] R. Pettit, J. Homer, R. Gee, A. Starbuck, and S. Mengel. 2015. An empirical study of iterative improvement in programming assignments. *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (2015), 410–415. DOI : https://doi.org/10.1145/2676723.2677279

[150] Reinhold Plösch and Cornelia Neumüller. 2020. Does Static Analysis Help Software Engineering Students?. In *Proceedings of the 2020 9th International Conference on Educational and Information Technology (ICEIT 2020)*. Association for Computing Machinery, New York, NY, USA, 247–253. DOI : https://doi.org/10.1145/3383923.3383957

[151] Bernard John Poole and Timothy S Meyer. 1996. Implementing a set of guidelines for CS majors in the production of program code. *ACM SIGCSE Bulletin* 28, 2 (1996), 43–48. DOI : https://doi.org/10.1145/228296.228304

[152] Simona Prokic, Katarina-Glorija Grujic, Nikola Luburic, Jelena Slivka, Aleksandar Kovacevic, Dragan Vidakovic, and Goran Sladic. 2021. Clean Code and Design Educational Tool. *International Convention on Information, Communication and Electronic Technology, MIPRO 2021 - Proceedings* (2021), 1601 – 1606. DOI : https://doi.org/10.23919/MIPRO52101.2021.9597196

[153] Lin Qiu and Christopher Riesbeck. 2008. An incremental model for developing educational critiquing systems: experiences with the Java Critiquer. *Journal of Interactive Learning Research* 19, 1 (2008), 119–145.

[154] Sarnath Ramnath and Brahma Dathan. 2008. Evolving an integrated curriculum for object-oriented analysis and design. In *Proc. of SIGCSE*. 337–341. DOI : https://doi.org/10.1145/1352322.1352252

[155] Michael J Rees. 1982. Automatic assessment aids for Pascal programs. *ACM Sigplan Notices* 17, 10 (1982), 33–42. DOI : https://doi.org/10.1145/948086.948088

[156] P. A. Relf. 2005. Tool assisted identifier naming for improved software readability: an empirical study. In *Symposium on Empirical Software Engineering*.

[157] S.S. Robinson and M.L. Soffa. 1980. An instructional aid for student programs. *ACM SIGCSE Bulletin* 12, 1 (1980), 118–129. DOI : https://doi.org/10.1145/953032.804623

[158] Gregorio Robles, Jesús Moreno-León, Efthimia Aivaloglou, and Felienne Hermans. 2017. Software clones in scratch projects: On the presence of copy-and-paste in computational thinking learning. In *IEEE 11th International Workshop on Software Clones (IWSC)*. IEEE, 1–7. DOI : https://doi.org/10.1109/iwsc.2017.7880506

[159] Jason Rogers and Chuck Pheatt. 2009. Integrating Antipatterns into the Computer Science Curriculum. *J. Comput. Sci. Coll.* 24, 5 (2009), 183–189.

[160] S. Rose, J. Habgood, and T. Jay. 2018. Pirate plunder: Game-based computational thinking using scratch blocks. *European Conference on Games-based Learning* (2018).

[161] S. Rose, J. Habgood, and T. Jay. 2019. Using Pirate Plunder to Develop Children's Abstraction Skills in Scratch. In *Proc. of Human Factors in Comp. Sys. (CHI)*. DOI : https://doi.org/10.1145/3290607.3312871

[162] S.P. Rose, M.P.J. Habgood, and T. Jay. 2020. Designing a Programming Game to Improve Children's Procedural Abstraction Skills in Scratch. *Journal of Educational Computing Research* 58, 7 (2020), 1372–1411. DOI : https://doi.org/10.1177/0735633120932871

[163] R.W. Roth. 1980. The teaching of documentation and good programming style in a liberal arts computer science program. *Proc. of SIGCSE* (1980). DOI : https://doi.org/10.1145/953032.804626

[164] I. B. Sampaio and L. Barbosa. 2016. Software readability practices and the importance of their teaching. In *Proc. of Information and Commun. Sys. (ICICS)*. DOI : https://doi.org/10.1109/iacs.2016.7476069

[165] Mincho Sandalski, Asya Stoyanova-Doycheva, Ivan Popchev, and Stanimir Stoyanov. 2011. Development of a refactoring learning environment. *Cybernetics and Information Technologies (CIT)* 11, 2 (2011).

[166] Dean Sanders and Janet Hartman. 1987. Assessing the quality of programs: A topic for the CS2 course. In *Proceedings of the eighteenth SIGCSE technical symposium on Computer science education*. 92–96. DOI : https://doi.org/10.1145/31726.31741

[167] Francisco Alan de O. Santos, Alana Oliveira, Carlos S. Soares Neto, and Mario Meireles Teixeira. 2021. Quality Assessment of Learners' Programs by Grouping Source Code Metrics. *International Conference on Computer Supported Education, CSEDU - Proceedings* 1 (2021), 339 – 346. DOI : https://doi.org/10.5220/0010457003390346

[168] L.P. Scatalon, J.C. Carver, R.E. Garcia, and E.F. Barbosa. 2019. Software testing in introductory programming courses: A systematic mapping study. In *SIGCSE*. DOI : https://doi.org/10.1145/3287324.3287384

[169] C. Schulte, T. Clear, A. Taherkhani, T. Busjahn, and J. H.wohli Paterson. 2010. An introduction to program comprehension for computer science educators. *ITiCSE working group reports* (2010), 65–86. DOI : https://doi.org/10.1145/1971681.1971687

[170] R. Sekimotot and K. Kaijirlt. 2000. A diagnosis system of programming styles using program patterns. *IEICE Transactions on Information and Systems* E83-D, 4 (2000), 722–728. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0033686202&partnerID=40&md5=53746e00c4515ce4fbbe353f773029f7

[171] A. Senger, S. H. Edwards, and M. Ellis. 2022. Helping Student Programmers Through Industrial-Strength Static Analysis: A Replication Study. *Proc. of SIGCSE* (2022), 8–14. DOI : https://doi.org/10.1145/3478431.3499310

[172] D. Silva, I. Nunes, and R. Terra. 2017. Investigating code quality tools in the context of software engineering education. *Computer Applications in Engineering Education* 25, 2 (2017), 230–241. DOI : https://doi.org/10.1002/cae.21793

[173] Dale Skrien. 2003. Learning appreciation for design patterns by doing it the hard way first. *Computer Science Education* 13, 4 (2003), 305–313. DOI : https://doi.org/10.1076/csed.13.4.305.17491

[174] S. Smith, S. Stoecklin, and C. Serino. 2007. An innovative approach to teaching refactoring. *SIGCSE* (2007).

[175] S.K. Sripada and Y.R. Reddy. 2015. Code Comprehension Activities in Undergraduate Software Engineering Course - A Case Study. In *Proc. of ASE*. DOI : https://doi.org/10.1109/aswec.2015.18

[176] M. Stegeman, E. Barendsen, and S. Smetsers. 2014. Towards an empirically validated model for assessment of code quality. *Koli Calling* (2014). DOI : https://doi.org/10.1145/2674683.2674702

[177] M. Stegeman, E. Barendsen, and S. Smetsers. 2016. Designing a rubric for feedback on code quality in programming courses. *Proc. of Koli Calling* (2016). DOI : https://doi.org/10.1145/2999541.2999555

[178] S. Stoecklin, S. Smith, and C. Serino. 2007. Teaching students to build well formed object-oriented methods through refactoring. *Proc. of SIGCSE* (2007). DOI : https://doi.org/10.1145/1227504.1227364

[179] S. Stoyanov, A. Stoyanova-Doycheva, I. Popchev, and M. Sandalski. 2011. ReLE - a refactoring supporting tool. *Comptes Rendus de L'Academie Bulgare des Sciences* 64, 7 (2011), 1017–1026. https://www.scopus.com/inward/record.uri?eid=2-s2.0-80052055171&partnerID=40&md5=073154e28cfc7142506ac67d4e668504

[180] S. Stuurman, H. Passier, and E. Barendsen. 2016. Analyzing students' software redesign strategies. *Koli Calling* (2016), 110–119. DOI : https://doi.org/10.1145/2999541.2999559

[181] P. Techapalokul and E. Tilevich. 2017. Enhancing block-based programming pedagogy to promote the culture of quality from the ground up a position paper. *Blocks and Beyond Workshop* (2017). DOI : https://doi.org/10.1109/blocks.2017.

8120420

[182] P. Techapalokul and E. Tilevich. 2017. Novice Programmers and Software Quality: Trends and Implications. *CSEET* (2017). DOI : https://doi.org/10.1109/cseet.2017.47

[183] P. Techapalokul and E. Tilevich. 2017. Understanding recurring quality problems and their impact on code sharing in block-based software. In *VL/HCC*. DOI : https://doi.org/10.1109/vlhcc.2017.8103449

[184] P. Techapalokul and E. Tilevich. 2019. Code Quality Improvement for All: Automated Refactoring for Scratch. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* 2019-October (2019), 117–125. DOI : https://doi.org/10.1109/VLHCC.2019.8818950

[185] E. Tempero and Y. Tu. 2021. Assessing Understanding of Maintainability using Code Review. *Proc. of ACE* (2021), 40–47. DOI : https://doi.org/10.1145/3441636.3442303

[186] T.K. Teodosiev and A.M. Nachev. 2015. Programming style in introductory programming courses. *International Journal of Applied Engineering Research* 10, 10 (2015), 26103–26114. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84932106986&partnerID=40&md5=ceec3aa75bea4ec5b6e9d2ca297ee3b4

[187] Veronika Thurner. 2019. Fostering the comprehension of the object-oriented programming paradigm by a virtual lab exercise. In *Experiment International Conference*. IEEE, 137–142. DOI : https://doi.org/10.1109/expat.2019.8876484

[188] Eli Tilevich, Peeratham Techapalokul, and Simin Hall. 2020. Teaching the culture of quality from the ground up: Novice-tailored quality improvement for scratch programmers.. In *In 2020 ASEE Annual Conference & Exposition*. ASEE Conferences.

[189] S. M. Tisha, R. A. Oregon, G. Baumgartner, F. Alegre, and J. Moreno. 2022. An Automatic Grading System for a High School-level Computational Thinking Course. *Proc. of the Workshop on Software Engineering Education for the Next Generation, SEENG* (2022), 20–27. DOI : https://doi.org/10.1145/3528231.3528357

[190] Graziela Simone Tonin, Alfredo Goldman, Carolyn Seaman, and Diogo Pina. 2017. Effects of technical debt awareness: A classroom study. In *International Conference on Agile Software Development*. Springer, Cham, 84–100. DOI : https://doi.org/10.1007/978-3-319-57633-6_6

[191] Nghi Truong, Paul Roe, and Peter Bancroft. 2004. Static Analysis of Students' Java Programs. In *Proceedings of the Sixth Australasian Conference on Computing Education (ACE '04)*. 317–325.

[192] L. C. Ureel II and C. Wallace. 2019. Automated Critique of Early Programming Antipatterns. In *Proc. of SIGCSE*. DOI : https://doi.org/10.1145/3287324.3287463

[193] Ángela Vargas-Alba, Giovanni Maria Troiano, Quinyu Chen, Casper Harteveld, and Gregorio Robles. 2019. Bad Smells in Scratch Projects: A Preliminary Analysis.. In *TACKLE@ EC-TEL*.

[194] A. Vasileva and D. Schmedding. 2017. How to improve code quality by measurement and refactoring. *Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016* (2017), 131–136. DOI : https://doi.org/10.1109/QUATIC.2016.034

[195] L.J. Waguespack. 2012. A design quality learning unit in relational data modeling based on thriving systems properties. *ISECON* 29 (2012).

[196] T. Wakabayashi, S. Ogata, and S. Matsuura. 2011. Dependency analysis for learning class structure for novice Java programmer. *IEEE 2nd International Conference on Software Engineering and Service Science* (2011), 532–535. DOI : https://doi.org/10.1109/ICSESS.2011.5982370

[197] Y.-Q. Wang, Z.-Y. Qi, L.-J. Zhang, and M.-J. Song. 2011. Research and practice on education of SQA at source code level. *International Journal of Engineering Education* 27, 1 PART 1 (2011), 70–76. https://www.scopus.com/inward/record.uri?eid=2-s2.0-79958804896&partnerID=40&md5=9bf92b1f075c79d5892b68d22eebbad5

[198] J. Whalley, T. Clear, P. Robbins, and E. Thompson. 2011. Salient elements in novice solutions to code writing problems. *Conferences in Research and Practice in Information Technology Series* 114 (2011), 37–45. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84872811419&partnerID=40&md5=34916932876877ba37667cb8a801f847

[199] E.S. Wiese, A.N. Rafferty, and A. Fox. 2019. Linking Code Readability, Structure, and Comprehension among Novices: It's Complicated. In *Proc. of ICSE-SEET*. DOI : https://doi.org/10.1109/icse-seet.2019.00017

[200] E.S. Wiese, A.N. Rafferty, D.M. Kopta, and J.M. Anderson. 2019. Replicating novices' struggles with coding style. *Proc. of ICPC* (2019). DOI : https://doi.org/10.1109/icpc.2019.00015

[201] Eliane Wiese, Anna N. Rafferty, and Jordan Pyper. 2022. Readable vs. Writable Code: A Survey of Intermediate Students' Structure Choices. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 321–327. DOI : https://doi.org/10.1145/3478431.3499413

[202] E.S. Wiese, M. Yen, A. Chen, L.A. Santos, and A. Fox. 2017. Teaching students to recognize and implement good coding style. *L@S 2017 - Proceedings of the 4th (2017) ACM Conference on Learning at Scale* (2017), 41–50. DOI : https://doi.org/10.1145/3051457.3051469

[203] C. Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proc. of Evaluation and Assessment in software engineering (EASE)s*. 1–10. DOI : https://doi.org/10.1145/2601248.2601268

[204] Wei Xu, Chao Wu, and Jianliang Lu. 2021. Exploration of Experimental Teaching Reforms on C Programming Design Course. *Proceedings - 2021 International Symposium on Advances in Informatics, Electronics and Education, ISAIEE 2021* (2021), 330 – 333. DOI : https://doi.org/10.1109/ISAIEE55071.2021.00086

[205] S. V. Yulianto and I. Liem. 2014. Automatic grader for programming assignment using source code analyzer. In *2014 International Conference on Data and Software Engineering (ICODSE)*. 1–4. DOI : https://doi.org/10.1109/ICODSE.2014.7062687

[206] M. Zaidman. 2004. Teaching Defensive Programming in Java. *J. Comput. Sci. Coll.* 19, 3 (2004).

[207] K. K. Zaw, H. W. Hnin, K. Y. Kyaw, and N. Funabiki. 2020. Software Quality Metrics Calculations for Java Programming Learning Assistant System. In *IEEE Conference on Computer Applications(ICCA)*. 1–6. DOI : https://doi.org/10.1109/ICCA49400.2020.9022823

[208] I. Zsigmond, M.I. Bocicor, and A.-J. Molnar. 2020. Gamification based learning environment for computer science students. *Proc. of Evaluation of Novel Approaches to SE (ENASE)* (2020). DOI : https://doi.org/10.5220/0009579305560563

Table 3: Complete list of papers and coding.
CS=code smells, DP=design patterns, MT=metrics, RD=readability, RF=refactoring, SA=static analysis

Title	Year	Topic	Method	Language	CS	DP	MT	RD	RF	SA
"I know it when i see it" - Perceptions of code quality, Börstler et al. [20]	2018	Perceptions	DescrCorr	generic				rd		
A card game for learning software-refactoring principles, Haendler [70]	2019	Materials	Survey	generic-OO	cs				rf	
A case study of the analysis of novice student programs, Mengel and Ulans [126]	1999	ProgramQuality	DescrCorr	C++			mt			sa
A case study of the static analysis of the quality of novice student programs, Mengel and Yerramilli [124]	1999	ProgramQuality	DescrCorr	C++			mt			sa
A critic for LISP, Fischer [52]	1987	SelfmadeTool	Tool	Lisp						
A design quality learning unit in OO modeling bridging the engineer and the artist, Waguespack [195]	2011	Curriculum	NoneUnclear	generic-OO			mt			
A diagnosis system of programming styles using program patterns, Sekimotot and Kaijirlt [170]	2000	SelfmadeTool	(Q)Experim	C				rd		
A framework for the assessment and training of software refactoring competences, Haendler and Neumann [71]	2019	Curriculum	CaseStudy	generic					rf	
A Heuristic Tool for Measuring Software Quality Using Program Language Standards, Abdallah and Alrifaee [1]	2022	SelfmadeTool	Tool	Java						
A Large-Scale Comparison of Python Code in Jupyter Notebooks and Scripts, Grotov et al. [68]	2022	ProgramQuality	DescrCorr	Python						
A method for detecting bad smells and ITS application to software engineering education, Ito et al. [85]	2014	SelfmadeTool	Tool	Java	cs				rf	
A pedagogical approach in interleaving software quality concerns at an artificial intelligence course, Cernau et al. [28]	2022	TeachingMethod	NoneUnclear	Java			mt			
A programming style taxonomy, Oman and Cook [141]	1991	Assignments	Other	generic						
A Proposal of Coding Rule Learning Function in Java Programming Learning Assistant System, Funabiki et al. [56]	2016	SelfmadeTool	Survey	Java				rd		sa
A Resource to Support Novices Refactoring Conditional Statements, Izu et al. [86]	2022	TeachingMethod	(Q)Experim	C	cs			rd	rf	
A study of loop style and abstraction in pedagogic practice, Barnes and Shinners-Kennedy [14]	2011	ProgramQuality	Qualitative	mutiple						
A study on the quality mindedness of students, Dick et al. [46]	2022	Perceptions	Survey	generic						
A tool for diagnosing the quality of java program and a method for its effective utilization in education, Hashiura et al. [76]	2010	ExternalTool	(Q)Experim	Java				rd		
A Tutoring System to Learn Code Refactoring, Keuning et al. [99]	2021	SelfmadeTool	Tool	Java					rf	
Academic coding guideline model - OCG, Gu and Dubey [69]	2014	Assignments	Discussion	C				rd		
Amelioration of Teaching Strategies by Exploring Code Quality and Submission Behavior, Bai et al. [11]	2019	Behaviour	DescrCorr	C++	cs					
An Agile classroom experience: Teaching TDD and refactoring, Carlson [26]	2008	ProgProcess	Experience	Java					rf	
An Applicability Study on Refactoring Principles in Reading-Based Programming Learning, Maeta and Matsumoto [120]	2022	Materials	(Q)Experim	Java		dp			rf	
An automated assessment system for analysis of coding convention violations in Java programming assignments, Chen et al. [31]	2018	SelfmadeTool	Tool	Java				rd		
An Automatic Grading System for a High School-level Computational Thinking Course, Tisha et al. [189]	2022	SelfmadeTool	QuantOther	Haskell						
An empirical study of COBOL programs via a style analyzer: The benefits of good programming style, Benander and Benander [16]	1989	Behaviour	DescrCorr	Cobol			mt			
An empirical study of iterative improvement in programming assignments, Pettit et al. [149]	2015	Behaviour	DescrCorr	C++			mt			
An empirical study on students' ability to comprehend design patterns, Chatzigeorgiou et al. [30]	2008	ConceptUnd	(Q)Experim	mutiple		dp	mt			
An Incremental Model for Developing Educational Critiquing Systems: Experiences with the Java Critiquer, Qiu and Riesbeck [153]	2008	SelfmadeTool	DescrCorr	Java						

Table 3: Complete list of papers and coding.
CS=code smells, DP=design patterns, MT=metrics, RD=readability, RF=refactoring, SA=static analysis

Title	Year	Topic	Method	Language	CS	DP	MT	RD	RF	SA
An innovative approach to teaching refactoring, Smith et al. [174]	2006	ProgProcess	NoneUnclear	generic-OO					rf	
An instructional aid for student programs, Robinson and Soffa [157]	1980	SelfmadeTool	DescrCorr	Fortran						
Analysis of Learning Behavior in an Automated Programming Assessment Environment: A Code Quality Perspective, Chen et al. [32]	2020	Behaviour	QuantOther	Java						
Analyzing students' software redesign strategies, Stuurman et al. [180]	2016	Behaviour	Qualitative	Java		dp			rf	
Anomaly Detection for Early Warning in Object-oriented Programming Course, Lu et al. [114]	2021	SelfmadeTool	Tool	Java						sa
Applying Code Quality Detection in Online Programming Judge, Liu and Woo [111]	2020	ExternalTool	DescrCorr	Python						
Applying gamification to motivate students to write high-quality code in programming assignments, Kasahara et al. [93]	2019	TeachingMethod	(Q)Experim	C			mt			
Are Undergraduate Creative Coders Clean Coders? A Correlation Study, Groeneveld et al. [67]	2022	ProgramQuality	DescrCorr	Java						
ASPA: A Static Analyser to Support Learning and Continuous Feedback on Programming Courses. An Empirical Validation, Luukkainen et al. [118]	2022	SelfmadeTool	Survey	Python						sa
Assessing software quality of agile student projects by data-mining software repositories, Koetter et al. [106]	2019	ProgramQuality	DescrCorr	unknown			mt			
Assessing the quality of programs: A topic for the CS2 course, Sanders and Hartman [166]	1987	ProgramQuality	Discussion	generic						
Assessing Understanding of Maintainability using Code Review, Tempero and Tu [185]	2021	Assignments	(Q)Experim	generic						
Automated critique of early programming antipatterns, Ureel II and Wallace [192]	2019	SelfmadeTool	Tool	Java						sa
Automatic analysis of functional program style, Michaelson [127]	1996	SelfmadeTool	Tool	SML						
Automatic assessment aids for Pascal programs, Rees [155]	1982	SelfmadeTool	DescrCorr	Pascal						
Automatic Assessment of the Design Quality of Student Python and Java Programs, Orr [142]	2022	SelfmadeTool	(Q)Experim	mutiple				rd		
Automatic detection of bad programming habits in scratch, Moreno and Robles [130]	2014	ProgramQuality	DescrCorr	Scratch						
Automatic grader for programming assignment using source code analyzer, Yulianto and Liem [205]	2014	SelfmadeTool	DescrCorr	mutiple						
Automatic programming assessment, Hung et al. [82]	1993	ProgramQuality	DescrCorr	Pascal			mt			
AutoStyle: Toward coding style feedback at scale, Choudhury et al. [35]	2015	SelfmadeTool	Tool	mutiple						
Bad Smells in Scratch Projects: A Preliminary Analysis, Vargas-Alba et al. [193]	2019	ProgramQuality	DescrCorr	Scratch	cs					
Beautiful JavaScript: How to guide students to create good and elegant code, Passier et al. [145]	2014	ProgProcess	NoneUnclear	JavaScript					rf	
Beauty and the Beast: on the readability of object-oriented example programs, Börstler et al. [19]	2016	Materials	DescrCorr	Java				rd		
But my program runs! Discourse rules for novice programmers, Joni and Soloway [90]	1986	Assignments	NoneUnclear	Pascal						
Can students help themselves? An investigation of students' feedback on the quality of the source code, Andrade and Brunet [7]	2019	ProgramQuality	Survey	Python						
Carrot and Stick approaches revisited when managing Technical Debt in an educational context, Crespo et al. [37]	2021	TeachingMethod	(Q)Experim	Java			mt			
Challenges of knowledge component modeling: A software engineering case study, Luburic et al. [117]	2022	Materials	CaseStudy	generic					rf	
Clean Code - Delivering A Lightweight Course, Chirvase et al. [33]	2021	TeachingMethod	NoneUnclear	Java						
Clean Code and Design Educational Tool, Prokic et al. [152]	2021	SelfmadeTool	Tool	C#	cs			rd		
Clean Code Tutoring: Makings of a Foundation, Luburić et al. [116]	2022	SelfmadeTool	(Q)Experim	C#				rd	rf	

Table 3: Complete list of papers and coding.
CS=code smells, DP=design patterns, MT=metrics, RD=readability, RF=refactoring, SA=static analysis

Title	Year	Topic	Method	Language	CS	DP	MT	RD	RF	SA
Cleangame: Gamifying the identification of code smells, Dos Santos et al. [47]	2019	SelfmadeTool	(Q)Experim	Java	cs				rf	
Code Comprehension Activities in Undergraduate Software Engineering Course - A Case Study, Sripada and Reddy [175]	2015	Behaviour	DescrCorr	mutiple					rf	
Code Perfumes: Reporting Good Code to Encourage Learners, Obermüller et al. [138]	2021	ProgramQuality	DescrCorr	Scratch	cs					
Code Quality Defects Across Introductory Programming Topics, Effenberger and Pelánek [49]	2022	ProgramQuality	DescrCorr	Python						
Code Quality Improvement for All: Automated Refactoring for Scratch, Techapalokul and Tilevich [184]	2019	SelfmadeTool	(Q)Experim	Scratch	cs		mt		rf	
Code quality issues in student programs, Keuning et al. [96]	2017	ProgramQuality	DescrCorr	Java						
Code quality: Examining the efficacy of automated tools, Hooshangi and Dasgupta [79]	2017	ExternalTool	DescrCorr	Python			mt			
CompareCFG: Providing Visual Feedback on Code Quality Using Control Flow Graphs, Jiang et al. [89]	2020	SelfmadeTool	Tool	Java						
Comparison of software quality in the work of children and professional developers based on their classroom exercises, Balogh [12]	2015	ProgramQuality	DescrCorr	Java						
Comprehension and application of design patterns by novice software engineers, Lartigue and Chapman [108]	2018	ConceptUnd	(Q)Experim	Java		dp				
Dependency Analysis for Learning Class Structure for Novice Java Programmer, Wakabayashi et al. [196]	2011	TeachingMethod	CaseStudy	Java				rd	rf	
Design of e-activities for the learning of code refactoring tasks, Lopez et al. [113]	2014	TeachingMethod	Experience	generic					rf	
Design patterns in scientific software, Gardner [60]	2004	TeachingMethod	NoneUnclear	Java		dp			rf	
Designing a Programming Game to Improve Children's Procedural Abstraction Skills in Scratch, Rose et al. [162]	2020	SelfmadeTool	(Q)Experim	Scratch	cs					
Designing a rubric for feedback on code quality in programming courses, Stegeman et al. [177]	2016	Assignments	Other	generic						
Detecting and Addressing Design Smells in Novice Processing Programs, Fehnker and de Man [51]	2019	ProgramQuality	DescrCorr	Processing	cs				rf	sa
Development of a refactoring learning environment, Sandalski et al. [165]	2011	SelfmadeTool	Tool	Java					rf	
Do code smells hamper novice programming? A controlled experiment on Scratch programs, Hermans and Aivaloglou [77]	2016	ConceptUnd	(Q)Experim	Scratch	cs					
Documentation Standards for Beginning Students, Brewer [24]	1976	Assignments	NoneUnclear	mutiple				rd		
Does Static Analysis Help Software Engineering Students?, Plösch and Neumüller [150]	2020	ExternalTool	DescrCorr	Java						sa
Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking, Moreno-León et al. [131]	2015	SelfmadeTool	(Q)Experim	Scratch						
DrPython-WEB: A Tool to Help Teaching Well-Written Python Programs, Battistini et al. [15]	2022	SelfmadeTool	Tool	Python						
Earthworm - Automated decomposition suggestions, Garg and Keen [61]	2018	SelfmadeTool	Tool	Python					rf	sa
Effectively teaching coding standards in programming, Li and Prasad [109]	2005	Perceptions	Survey	generic						
Effects of technical debt awareness: A classroom study, Tonin et al. [190]	2017	TeachingMethod	Qualitative	generic						
Encapsulation and Reuse as Viewed by Java Students, Fleury [53]	2001	Perceptions	Qualitative	Java						
Enhancing Abstraction in App Inventor with Generic Event Handlers, Patton et al. [146]	2019	SelfmadeTool	Tool	APPInventor	cs				rf	
Enhancing block-based programming pedagogy to promote the culture of quality from the ground up - a position paper, Techapalokul and Tilevich [181]	2017	Curriculum	Discussion	generic-block						
Evaluating Code Improvements in Software Quality Course Projects, Chren et al. [36]	2022	TeachingMethod	(Q)Experim	Java						sa

14

Table 3: Complete list of papers and coding.
CS=code smells, DP=design patterns, MT=metrics, RD=readability, RF=refactoring, SA=static analysis

Title	Year	Topic	Method	Language	CS	DP	MT	RD	RF	SA
Evolving an integrated curriculum for object-oriented analysis and design, Ramnath and Dathan [154]	2008	Curriculum	Experience	generic-OO		dp			rf	
Exploration of Experimental Teaching Reforms on C Programming Design Course, Xu et al. [204]	2021	TeachingMethod	DescrCorr	C						
Exploring Metrics for the Analysis of Code Submissions in an Introductory Data Science Course, Nguyen et al. [134]	2021	ProgramQuality	DescrCorr	Python			mt			
Five reasons for including technical debt in the software engineering curriculum, Falessi and Kruchten [50]	2015	Curriculum	NoneUnclear	generic						
Foobaz: Variable name feedback for student code at scale, Glassman et al. [63]	2015	SelfmadeTool	Survey	Python						
Forming groups for collaborative learning in introductory computer programming courses based on students' programming styles: An empirical study, De Faria et al. [41]	2006	ProgProcess	(Q)Experim	C			mt			
Fostering the comprehension of the object-oriented programming paradigm by a virtual lab exercise, Thurner [187]	2019	TeachingMethod	Experience	Java						
FrenchPress gives students automated feedback on Java program flaws, Blau and Moss [18]	2015	SelfmadeTool	Survey	Java						
Function Names: Quantifying the Relationship Between Identifiers and Their Functionality to Improve Them, Charitsis et al. [29]	2022	SelfmadeTool	Tool	Java				rd		
Gamification based learning environment for computer science students, Zsigmond et al. [208]	2020	SelfmadeTool	Tool	mutiple						sa
Helping Student Programmers Through Industrial-Strength Static Analysis: A Replication Study, Senger et al. [171]	2022	Behaviour	DescrCorr	Java						sa
High School Teachers' Understanding of Code Style, Kirk et al. [105]	2020	Perceptions	Qualitative	generic						
How junior developers deal with their technical debt?, Gilson et al. [62]	2020	Behaviour	Mixed	mutiple						sa
How kids code and how we know: An exploratory study on the scratch repository, Aivaloglou and Hermans [4]	2016	ProgramQuality	DescrCorr	Scratch	cs					
How teachers would help students to improve their code, Keuning et al. [97]	2019	ProgramQuality	Survey	generic						
How to improve code quality by measurement and refactoring, Vasileva and Schmedding [194]	2016	ProgProcess	DescrCorr	Java			mt		rf	sa
Human vs. Automated coding style grading in computing education, Perretta et al. [147]	2019	ProgramQuality	DescrCorr	C++						sa
Hyperstyle: A Tool for Assessing the Code Quality of Solutions to Programming Assignments, Birillo et al. [17]	2022	SelfmadeTool	Tool	mutiple				rd	rf	
Impact of aspect-oriented programming on the quality of novices' programs: A comparative study, Katić et al. [94]	2013	ProgramQuality	(Q)Experim	C#			mt	rd		
Implementing a set of guidelines for CS majors in the production of program code, Poole and Meyer [151]	1996	Assignments	Survey	Modula2						
Improving Feedback on GitHub Pull Requests: A Bots Approach, Hu and Gehringer [81]	2019	SelfmadeTool	Mixed	generic-OO	cs					sa
Improving Readability of Scratch Programs with Search-based Refactoring, Adler et al. [3]	2021	SelfmadeTool	Tool	Scratch				rd	rf	
Improving students programming quality with the continuous inspection process: a social coding perspective, Lu et al. [115]	2019	ProgProcess	(Q)Experim	Java						
Improving the software quality - An educational approach, Bozhikova et al. [22]	2017	SelfmadeTool	Tool	C#		dp			rf	
Integrating Antipatterns into the Computer Science Curriculum, Rogers and Pheatt [159]	2009	Curriculum	NoneUnclear	generic-OO		dp			rf	
Investigating code quality tools in the context of software engineering education, Silva et al. [172]	2017	ExternalTool	DescrCorr	Java			mt		rf	
Investigating static analysis errors in student Java programs, Edwards et al. [48]	2017	ProgramQuality	DescrCorr	Java						sa

Table 3: Complete list of papers and coding.
CS=code smells, DP=design patterns, MT=metrics, RD=readability, RF=refactoring, SA=static analysis

Title	Year	Topic	Method	Language	CS	DP	MT	RD	RF	SA
Investigation of the relationship between program correctness and programming style, Grigas [66]	1999	ProgramQuality	DescrCorr	mutiple			mt	rd		
Japroch: A tool for checking programming style, Mäkelä and Leppänen [121]	2004	SelfmadeTool	Tool	Java						
JMetricGrader: A software for evaluating student projects using design object-oriented metrics and neural networks, Celosmanovic and Ljubovic [27]	2022	ProgramQuality	QuantOther	Java			mt			
Learning appreciation for design patterns by doing it the hard way first, Skrien [173]	2003	TeachingMethod	Experience	Java		dp			rf	
Learning software engineering principles using open source software, Nandigam et al. [132]	2008	Assignments	NoneUnclear	Java			mt	rd	rf	
Learning to listen for design, Baniassad et al. [13]	2019	ProgProcess	Discussion	generic	cs	dp			rf	
Linking code readability, structure, and comprehension among novices: It's complicated, Wiese et al. [199]	2019	Perceptions	Survey	mutiple				rd		
Litterbox: A linter for scratch programs, Fraser et al. [55]	2021	SelfmadeTool	Tool	Scratch	cs					
Measuring static quality of student code, Breuker et al. [23]	2011	ProgramQuality	DescrCorr	Java			mt			
Measuring students' source code quality in software development projects through commit-impact analysis, Hamer et al. [75]	2021	Behaviour	DescrCorr	mutiple			mt			
Mind the Gap: Searching for Clarity in NCEA, Kirk et al. [103]	2021	Materials	Mixed	generic						
Mining student CVS repositories for performance indicators, Mierle et al. [128]	2005	Behaviour	DescrCorr	mutiple						
Modeling Learners Programming Skills and Question Levels Through Machine Learning, Kim et al. [101]	2020	ProgramQuality	QuantOther	mutiple				rd		
Novice Programmers and Software Quality: Trends and Implications, Techapalokul and Tilevich [182]	2017	ProgramQuality	DescrCorr	Scratch	cs					
On assuring learning about code quality, Kirk et al. [102]	2020	Curriculum	CaseStudy	generic						
On the Use of FCA Models in Static Analysis Tools to Detect Common Errors in Programming, Cristea et al. [38]	2021	ProgramQuality	DescrCorr	Python						sa
Pirate plunder: Game-based computational thinking using scratch blocks, Rose et al. [160]	2018	SelfmadeTool	Tool	Scratch	cs					
Program decomposition and complexity in CS1, Keen and Mammen [95]	2015	TeachingMethod	(Q)Experim	C			mt			
Programming style in introductory programming courses, Teodosiev and Nachev [186]	2015	Curriculum	NoneUnclear	generic						
Promoting Code Quality via Automated Feedback on Student Submissions, Karnalim and Simon [91]	2021	SelfmadeTool	Tool	mutiple						
Qualitative aspects of students' programs: Can we make them measurable?, Araujo et al. [8]	2016	SelfmadeTool	(Q)Experim	Python						
Quality Assessment of Learners' Programs by Grouping Source Code Metrics, Santos et al. [167]	2021	ProgramQuality	QuantOther	Lua			mt			
Readable vs. Writable Code: A Survey of Intermediate Students' Structure Choices, Wiese et al. [201]	2022	Perceptions	Survey	Java				rd		
RefacTutor: An Interactive Tutoring System for Software Refactoring, Haendler et al. [73]	2020	SelfmadeTool	Tool	Java					rf	
Reflections on teaching refactoring: A tale of two projects, Abid et al. [2]	2015	ProgProcess	(Q)Experim	Java						
ReLE - a refactoring supporting tool, Stoyanov et al. [179]	2011	SelfmadeTool	Tool	Java					rf	
Replicating novices' struggles with coding style, Wiese et al. [200]	2019	Perceptions	Survey	mutiple				rd		
Research and practice on education of SQA at source code level, Wang et al. [197]	2011	TeachingMethod	CaseStudy	generic						
Salient elements in novice solutions to code writing problems, Whalley et al. [198]	2011	ProgramQuality	Qualitative	mutiple						
Scale-driven automatic hint generation for coding style, Choudhury et al. [34]	2016	SelfmadeTool	(Q)Experim	mutiple						

Table 3: Complete list of papers and coding.
CS=code smells, DP=design patterns, MT=metrics, RD=readability, RF=refactoring, SA=static analysis

Title	Year	Topic	Method	Language	CS	DP	MT	RD	RF	SA
Serious refactoring games, Haendler and Neumann [72]	2019	Materials	NoneUnclear	generic-OO	cs				rf	
Smells in block-based programming languages, Hermans et al. [78]	2016	ProgramQuality	DescrCorr	generic-block	cs					
Software analytics to support students in object-oriented programming tasks: an empirical study, Ardimento et al. [9]	2020	ProgramQuality	(Q)Experim	Java						
Software clones in scratch projects: On the presence of copy-and-paste in computational thinking learning, Robles et al. [158]	2017	ProgramQuality	DescrCorr	Scratch						
Software engineer education support system ALECSS utilizing devOps tools, Ohtsuki et al. [140]	2016	SelfmadeTool	DescrCorr	Java						sa
Software metrics as a programming training tool, Bowman and Newman [21]	1990	Assignments	(Q)Experim	Cobol			mt			
Software Quality as a Subsidy for Teaching Programming, Gomes et al. [64]	2021	TeachingMethod	DescrCorr	Java						
Software Quality Metrics Calculations for Java Programming Learning Assistant System, Zaw et al. [207]	2020	SelfmadeTool	DescrCorr	Java			mt			
Software readability practices and the importance of their teaching, Sampaio and Barbosa [164]	2016	Assignments	Survey	generic-OO	cs			rd		
Sprinter: A Didactic Linter for Structured Programming, Alfredo et al. [6]	2022	SelfmadeTool	Tool	Java						
Static analyses in python programming courses, Liu and Petersen [110]	2019	SelfmadeTool	(Q)Experim	Python						sa
Static analysis of programming exercises: Fairness, usefulness and a method for application, Nutbrown and Higgins [137]	2016	ExternalTool	DescrCorr	Java						sa
Static analysis of source code written by novice programmers, Delev and Gjorgjevikj [44]	2017	ExternalTool	DescrCorr	C						sa
Static Analysis of Students' Java Programs, Truong et al. [191]	2004	SelfmadeTool	Tool	Java			mt			sa
Structural analysis of source code collected from programming contests, Park et al. [144]	2014	ProgramQuality	QuantOther	C++						
Student Refactoring Behaviour in a Programming Tutor, Keuning et al. [98]	2020	Behaviour	DescrCorr	Java					rf	
Students Projects' Source Code Changes Impact on Software Quality Through Static Analysis, Hamer et al. [74]	2021	Behaviour	DescrCorr	mutiple						sa
Studying Software Metrics Based on Real-World Software Systems, Liu et al. [112]	2007	Assignments	NoneUnclear	generic			mt			
Supporting Students in C++ Programming Courses with Automatic Program Style Assessment, Ala-Mutka et al. [5]	2004	SelfmadeTool	Qualitative	C++						
Teacher Mate: A Support Tool for Teaching Code Quality, de Araújo et al. [40]	2020	SelfmadeTool	DescrCorr	Java						
Teaching code quality in high school programming courses - Understanding teachers' needs, Kirk et al. [104]	2022	Perceptions	Qualitative	generic						
Teaching Defensive Programming in Java, Zaidman [206]	2004	Assignments	Survey	Java						
Teaching design patterns using a family of games, Gómez-Martín et al. [65]	2009	Assignments	Experience	Java		dp			rf	
Teaching programming style with ugly code, McMaster et al. [123]	2013	SelfmadeTool	Tool	Java				rd		
Teaching software quality via source code inspection tool, de Andrade Gomes et al. [39]	2017	SelfmadeTool	(Q)Experim	mutiple						
Teaching students to build well formed object-oriented methods through refactoring, Stoecklin et al. [178]	2007	ProgProcess	NoneUnclear	generic-OO					rf	
Teaching students to recognize and implement good coding style, Wiese et al. [202]	2017	ProgramQuality	(Q)Experim	Python						
Teaching the culture of quality from the ground up: Novice-tailored quality improvement for scratch programmers, Tilevich et al. [188]	2020	SelfmadeTool	Mixed	Scratch					rf	
The effect of reporting Known issues on students' work, Gaber and Kirsh [57]	2018	Perceptions	(Q)Experim	C++						
The Five Million Piece Puzzle: Finding Answers in 500,000 Snap-Projects, Jatzlau et al. [88]	2019	ProgramQuality	DescrCorr	Snap!	cs					

Table 3: Complete list of papers and coding.
CS=code smells, DP=design patterns, MT=metrics, RD=readability, RF=refactoring, SA=static analysis

Title	Year	Topic	Method	Language	CS	DP	MT	RD	RF	SA
The impact of automated code quality feedback in programming education, Jansen et al. [87]	2017	ExternalTool	(Q)Experim	mutiple						
The LAN-simulation: A refactoring teaching example, Demeyer et al. [45]	2005	Assignments	Experience	Java					rf	
The Role of Source Code Vocabulary in Programming Teaching and Learning, Nascimento et al. [133]	2020	SelfmadeTool	(Q)Experim	Python				rd		
The teaching of documentation and good programming style in a liberal arts computer science program, Roth [163]	1980	TeachingMethod	NoneUnclear	Basic						
Tool assisted identifier naming for improved software readability: An empirical study, Relf [156]	2005	SelfmadeTool	(Q)Experim	Java				rd		
Towards an empirically validated model for assessment of code quality, Stegeman et al. [176]	2014	Assignments	Qualitative	generic						
Towards generalizing expert programmers' suggestions for novice programmers, Ichinco et al. [83]	2013	ProgramQuality	Survey	Alice-LG						sa
Understanding recurring quality problems and their impact on code sharing in block-based software, Techapalokul and Tilevich [183]	2017	ProgramQuality	DescrCorr	Scratch	cs					
Understanding Refactoring Tasks over Time: A Study Using Refactoring Graphs, Brito et al. [25]	2022	Behaviour	(Q)Experim	Java					rf	
Understanding Semantic Style by Analysing Student Code, De Ruvo et al. [43]	2018	ProgramQuality	DescrCorr	Java						
Unencapsulated collection - A teachable design smell, De Ruvo et al. [42]	2018	ProgramQuality	CaseStudy	generic-OO	cs				rf	
Unreadable code in novice developers, Avila et al. [10]	2021	Perceptions	Survey	generic				rd		
Using examples as guideposts for programming exercises: Providing support while preserving the challenge, Gaber and Kirsh [58]	2021	Assignments	CaseStudy	C++					rf	
Using pirate plunder to develop children's abstraction skills in scratch, Rose et al. [161]	2019	SelfmadeTool	(Q)Experim	Scratch	cs					
Using project-based approach to teach design patterns: An Experience Report, Karre et al. [92]	2021	TeachingMethod	(Q)Experim	Java	cs	dp			rf	
Using software metrics tools for maintenance decisions: a classroom exercise, Marshall et al. [122]	1996	ExternalTool	CaseStudy	unknown			mt			
Using static analysis tools to assist student project evaluation, Molnar et al. [129]	2020	ExternalTool	DescrCorr	Python						sa
Using Verilog LOGISCOPE to analyze student programs, Mengel and Ulans [125]	1998	ExternalTool	DescrCorr	C++						sa
Utilizing software engineering education support system ALECSS at an actual software development experiment: A case study, Ohtsuki and Kakeshita [139]	2019	SelfmadeTool	DescrCorr	Java						
You have said too much : Java-like verbosity anti-patterns in python codebases, Ma and Tilevich [119]	2021	ProgramQuality	DescrCorr	Python						