Software Quality as a Subsidy for Teaching Programming

Pedro Henrique Gomes
Faculty of Sciences and Technology
São Paulo State University (Unesp)
Presidente Prudente-SP, Brazil
pedro.gomes@unesp.br

Rogério Eduardo Garcia
Faculty of Sciences and Technology
São Paulo State University (Unesp)
Presidente Prudente-SP, Brazil
rogerio.garcia@unesp.br

Danilo Medeiros Eler
Faculty of Sciences and Technology
São Paulo State University (Unesp)
Presidente Prudente-SP, Brazil
danilo.eler@unesp.br

Ronaldo Celso Correia
Faculty of Sciences and Technology
São Paulo State University (Unesp)
Presidente Prudente-SP, Brazil
ronaldo.correia@unesp.br

Celso Olivete Junior
Faculty of Sciences and Technology
São Paulo State University (Unesp)
Presidente Prudente-SP, Brazil
celso.olivete@unesp.br

***Abstract*—Well-written code (which meets standards, conventions, and quality specifications) is often associated with the programmer's experience, which is why companies have been looking for increasingly qualified professionals. Codes at odds with these standards tend to be complex and poorly written and are generally difficult to understand. Consequently, the activities inherent to it become costly. Although well defined and consolidated in the industry, the concepts of code quality taught in the academy are insufficient to enable students to meet market expectations. Research indicates that graduates do not feel prepared to enter the labor market, facing difficulties when competing for the best job opportunities. This difficulty highlights a gap between industry and academia. Researchers have identified the gap and have proposed improvements to the teaching-learning process based on using the concepts and tools widely used in the software industry in an academic environment. When analyzing educational institutions' programming disciplines, it is possible to notice a mismatch between programming disciplines and code quality disciplines. In this scenario, this project aims to propose improvements on how knowledge of programming and quality is evolved, proposing an approach that uses software quality as a subsidy for teaching programming, providing the teacher with guidelines for teaching programming focusing on internal quality source code. We idealized an innovative practice that brought programming, usually focused on the execution of algorithms, within the teaching of code quality, focused on software engineering principles. The use of code inspection tools allowed direct teaching according to the class's needs, introducing a guided content-based internal quality teaching in programming disciplines without generating extra work for the teacher. In addition to the improvements for the teacher, we are convinced that from the student's perspective, we were able to motivate them to learn to program not only concerned with the execution results but also how the solution is developed.**

***Index Terms*—Course assessment, Professional development**

I. Introduction

Source Code quality is related to comprehensibility. It is possible to associate it with the following question: how easy is it for other programmers to understand a code snippet, and how well can it be extended and reused? [1], [2]

Teaching Programming Classes do not focus on source code quality, except when associated with the teacher's objectives, which address concepts beyond the subject program, proposed in pedagogical projects approved by teachers' councils.

A. Context and Motivation

Studies suggest that teaching software quality should begin as early as possible [3]–[5]. Recent works propose the adoption of Source Code Inspection Tools on teaching environment aims to provide subside for students on understanding not to write dirty code [6], [7] and for the teacher to conduct the subject based on explicit difficulties [7], [8].

We recently proposed an approach to support source code analysis to identify students' lack of good programming practices. The proposed tool uses a source code analysis tool for static code inspection to identify the student's violations.

Although the approach allowed the teacher to better understand the students' evolution, we consider feedback received by local users and reviewers in recent publications.

Significant notes stating the understanding of what a given violation means in the context of a subject is complicated, both to the teacher and the student's perspective. So we decided to relate each of the most violated rules to the concepts of the discipline.

We provided the teacher a better understanding of the subject level, so we proposed guidelines to assist the same in conducting the discipline. In parallel, we provide the student with a set of explanations and practical examples within the scope of learning.

B. The Problem

As explained, it is possible to fragment the problem from the perspective of the teaching-learning process, of the teacher and the student. In this scenario, the problem consists of:

We are grateful to FAPESP – Sao Paulo Research Foundation and CAPES (CAPES - DS) for the financial support.

1) introduce internal (source code) quality teaching in programming subjects;
2) the extra effort needed by the teacher for this introduction to take effect;
3) students have no motivation to worry about their source code quality during the programming classes;

This paper presents *Teacher Mate (TM)* tool New Module, with improvements based on teacher and academic community feedback [7].

II. Background

A. Code Quality

The task of identifying software with poor code quality is not always trivial. The average user may not even understand what coding is, but there are several reasons why code quality is essential. Over time, the user may face situations in that software performs unexpected behavior, from slowness to situations of total inoperability with risk of loss of information.

The initial cost of developing quality software can be daunting, but studies show that poor quality code begins to affect the project's development in a short time. According to the developers themselves, that moment can occur in a few weeks [9].

Many authors have focused on understanding how poorly structured, non-standard, and dirty-code with programmer addictions directly impact software development. These studies show how important it is to follow standards at the time of development and propose alternatives to improve and perfect what is being produced [1], [2], [9], [10].

Regarding the relation between poorly written code and the lack of developer experience, the industry looks for professionals who have the skills of a good code writer.

B. Market Expectation

Studies point out companies' difficulties in finding professionals who meet the requirements of the available job offers, as well as the difficulty of recent graduates to fulfill requirements and, consequently, become effective in the market [11]. In this sense, studies pointed to a gap between the profile of graduated and expected by the companies [11]–[13].

C. Teaching Context

This gap was also identified in the academy context [14], [15]. The authors highlighted that good practices could promote critical thinking while learning programming concepts. The application of code quality could help students during the course and after graduating, reducing the gap between academia and industry.

Although students recently enrolled in the university are encouraged to improve the internal quality of their developed programs. However, students are introduced to the concepts of Software Engineering and taught fundamental concepts such as Quality Assurance [5], [6], [8] only in later years.

For this reason, we chose to use a set of validated support tools, aiming at reducing this gap by advancing the student's exposure to concepts related to subsequent years of the course, like source code internal quality.

III. Support Tools

Support tools are considered the tools that supported the implementation process and their respective uses within the scope of the problem, not being considered functionalities unrelated to the project. In this scenario, version control tools, integrated development environments, debuggers, browsers, and development frameworks are disregarded.

A. SonarQube™ Platform

The SonarQube™ (SQ) platform (Figure 1) is a continuous source code inspection tool widely used by the software industry. The adoption of tools such as SQ within a teaching environment has been shown to be successful in several studies [6]–[8], [16].

Although SQ is helpful to analyze a project, it was not prepared to compare projects. It is important to note that SQ does not analyze multiples source codes at a time. This limitation could be considered a weakness when adopting SQ for educational purposes. It is difficult to compare multiple project reports, becoming an arduous task to analyze and compare the reports of each project to identify violations of the students' source code [6].

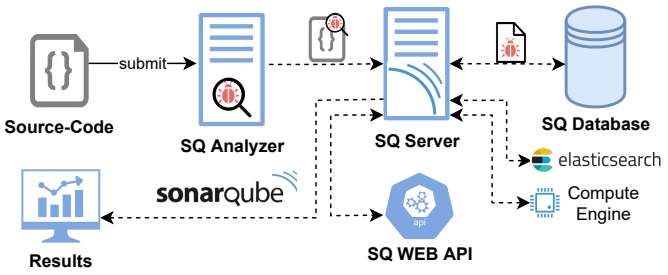

Fig. 1. Representation of SonarQube Architecture. Based on [17].

However, as one of the challenges of the work is to ensure that the teacher does not have an increase in the workload, it was necessary to provide another environment for interaction, supported by the *Teacher Mate* tool, explained below.

B. Teacher Mate Tool

Teacher Mate focus on supporting the teacher and helping in identifying students' difficulties in acquiring good programming practices by identifying code snippets that violate programming conventions and standards. This tool implements an approach that allows the teacher to create classes and group the quality reports generated by the SQ tool. This combination solves one of the difficulties pointed out in using SQ for educational purposes, allowing the comparison of several code snippets produced by different students.

One may see in Figure 2, *Teacher Mate* tool uses the source code inspection reports provided by the SonarQube Scanner. These reports are made available via the SonarQube WEB API, making *Teacher Mate* tool possible to provide its reports in addition to a series of visualizations to support both the teacher (in terms of grouped activities) and the student (viewing your submission report).

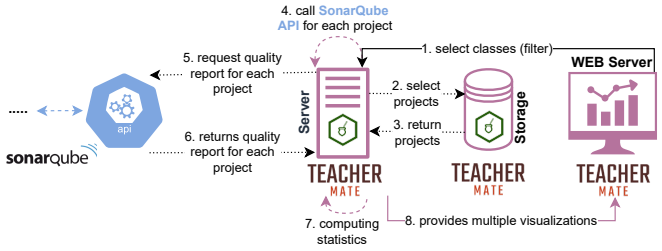

Fig. 2. Representation of *Teacher Mate* Architecture and Integration with SonarQube. Adopted from [7].

In summary, the *Teacher Mate* tool focused on two challenges: 1) explaining the students' difficulties in following good programming practices; 2) enabling the teacher to monitor the progress of the class as a function of time, through the possibility of grouping and comparing (Figure 3).

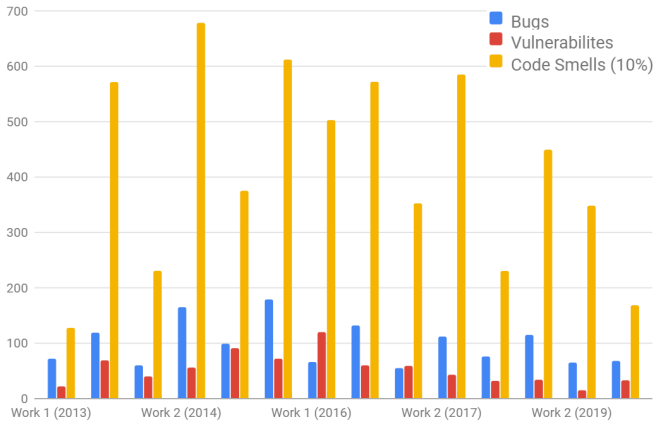

Fig. 3. Example of *Teacher Mate* Report, considering classes evolution.

IV. Artifacts

In this section, we present the I/O artifacts. All conceptual, physical, or logical items used are considered artifacts, whether from the adoption of existing and validated (inputted) or the development of new (outputted) artifacts.

Figure 4 presents a schema between artifacts related to this methodology. **Bold text** items are considered adopted artifacts, while highlighted texts are produced ones. Is it possible to notice that some artifacts are both **bolded and highlighted**. In this case, the artifact was adopted, but according to criteria explained in the following.

A. Inputted Artifacts

1) ***SonarQube Violation Database****:* This repository contains the analysis of 180 projects developed by students as requirements for obtaining credits in the Object-Oriented Programming (OOP) subject. The dataset was created by collecting data from students' source code, considering classes from 2013 to 2019, withing more than 60 thousand violations, as shown in Table I.

TABLE I
Ammount of violations identified, listed by year classes and violation class.

Year Classes	Violations			
	Bug	***Vulnerability***	***Code-Smells***	***Total***
2013	193	93	7338	*7624*
2014	227	98	9981	*10306*
2015	280	165	10744	*11189*
2016	200	182	11302	*11684*
2017	169	104	10513	*10786*
2018	193	68	6941	*7202*
2019	135	50	5203	*5388*
Total	*1397*	*760*	*62022*	***64179***

Selection Process: it was necessary to define an instance of the violations repository (Figure 4) in order to allow the teacher to define the guidelines for a specific class. It contains only violations related to the projects that are part of the classes defined by the teacher. This instance is generated at execution time, according to teacher's input, being used to group Linked Rules List (IV-B2), compute Guidelines (IV-B3) and related visualizations.

2) ***SonarQube Violated Rules List****:* To identify code snippets that odds with conventions and standardization, it became necessary to define which analysis criteria would be adopted. As presented on II-C, many studies demonstrated that the use of CI tools in an educational environment proved to be a good approach. In this project's scope, we chose to use the rules provided by the SonarQube™ platform.

Selection Process: by analyzing the Figure 4, the Violated Rules List generates another group. Altogether an amount of 430 rules, utilized by SQ and maintained by SonarSource[1] composes the default analysis profile of SonarQube Scanner.

Regarding all the violations, only 136 rules are violated. Since all submissions, from all students from the year 2013 to 2019, only 37 rules had at least 1% of occurrences. Regarding the negligible impact on the proposal, in addition to the significant time spent. Rules with less than 1% of occurrence were disregarded in the following analyzes.

3) ***SonarQube Conformity Rules List****:* All SonarQube Rules are related to some standardization. **SonarQube Conformity Rules List** shows why the violation should not be committed, providing theoretical support to the teacher, students, and author(during the development of the proposed methodology).

When accessing the associated content available, the user (teacher or student) will encounter more technical content, which includes: examples of compliant and non-compliant code; associated risk, which includes the degree of severity and cost of remediation; In some cases, there are discussions promoted by voluntary participants that allow the user to post their doubts or suggestions, thus contributing to the maturity of the individual and the projects involved.

[1]Available at: https://rules.sonarsource.com/java

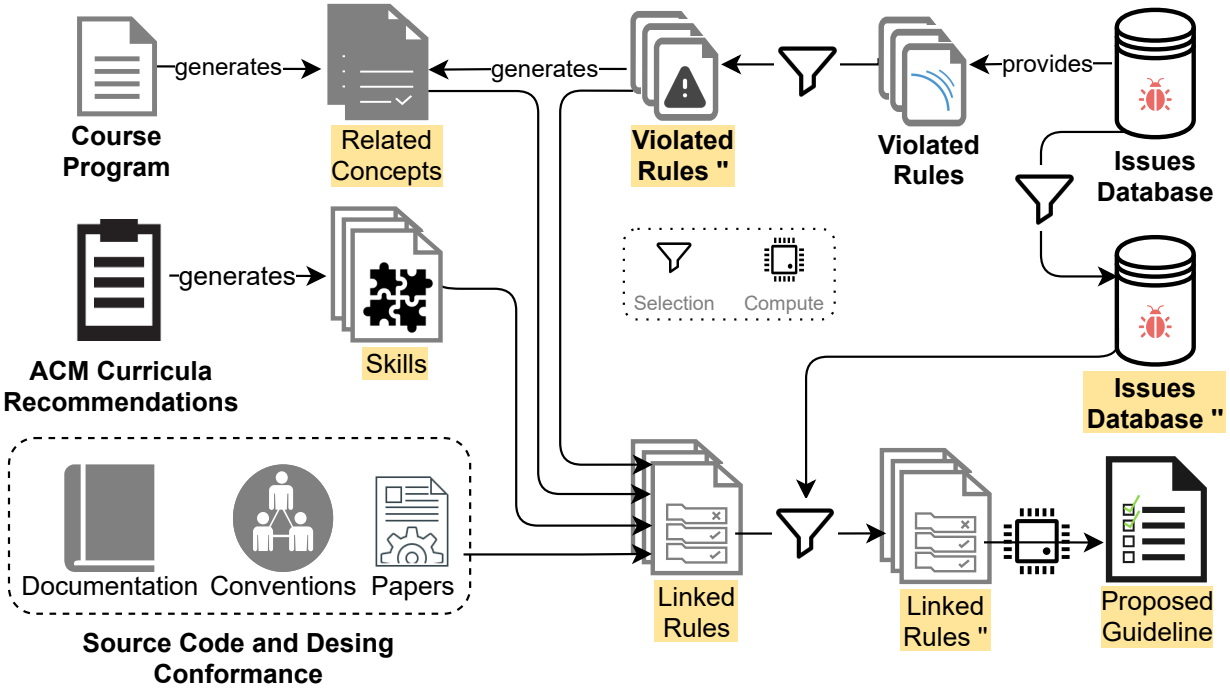

Fig. 4. Representation of artifacts related to this methodology.

Each rule can be associated with more than one conformity, and each conformity may contain more than one rule although, we define to associate only one conformity with each rule. The map of conventions related to SonarQube Rules could be seen in Figure 5.

*4) **Violation Related Skills List**:* Competences (or Skills) are capabilities evolved by students during academic courses. We identify the ordinary skills of the student during the subject by analyzing Curricula Recommendations like [18], [19].

Skills related to System Development capabilities were analyzed, resulting in the following skills (e.g.):

- Architecture & Design – creational (*Constructors, Singletons*), structural (*MVC, MVVM*) and behavioral (*Responsibility Attribution*) patterns;
- System Development – code conventions, such as nomenclature, alignment and indentation, coverage and unit tests, declarations, use of comments and code documentation;
- Techniques and Tools – specific resources provided by the Programming Language adopted in class. Within the scope of this project, the rules for correct adoption of techniques and tools considered Java Platform[2], such as the use of *Collections*, *Iterators* and *Packages*.

B. Outputted Artifacts

The artifacts necessary for applying the proposed methodology – designed and developed by the authors – are presented below. Being considered output artifacts since all of these artifacts were made available in an open-access repository[3].

*1) **Rules' Related Concepts List**:* Related Concepts are contents taught in disciplines, which allow the student to develop skills in a specific competence. These contents are commonly identified in the syllabus of the subjects and the lesson plans. This model follows the idea presented on II-C.

An issue can result from a lack of knowledge or skill in multiple concepts, so it is possible to link several concepts with the same rule and several rules.

*2) **Linked Rules List**:* The following are examples of evaluated rules that can compose the Linked Rules repository. The rules were arbitrarily selected in order to facilitate the reader to exemplify how the guidelines are generated.

***R1 – Resources should be closed**:*

Definition: Conections, streams, files and classes that implements *Closable*, must be closed after use.

Problem: the lack of resources closure implies a memory leak.

Related Concepts: Garbage Collector, Execution Flow, Constructors and Destructors, Data Input/Output, Exceptions and Throwable.

Code Samples:

Non-compliance

```
private void doSomething() {
  OutputStream stream = null;
  try {
```

[2]Available at: https://www.oracle.com/java/

[3]Available at http://www2.fct.unesp.br/grupos/lapesa/teachermate

Fig. 5. Map of Conformity related With SonarQube Rules List.

```
      for (String prop : propList) {
        st = new FileOutputStream("f");
      /* ... */
      }
    } catch (Exception e) {
      /* ... */
    } finally {
      st.close();
     //Multiple streams were opened.
     //Only the last is closed.
    }
}
```

Compliance

```
private void doSomething() {
  OutputStream stream = null;
  try {
    st = new FileOutputStream("f");
    for (String prop : propList) {
    /* ... */
    }
 } catch (Exception e) {
    /* ... */
 } finally {
    st.close();
  }
}
```

Proposed Solution: Use the *finally* block or *try-with-resources* standard.

Conformity: MITRE, CWE-459 - Incomplete Cleanup

Skill: Techniques and Tools

R2 – Standard outputs should not be used directly to log anything:

Definition: The use of standards output to register logs is highly inadvisable.

Problem: Log records must be easily retrievable, must have a uniform structure and must restrict access when necessary.

Related Concepts: Data Input/Output, Loggers, StackTrace

Code Samples:

Non-compliance

```
private void doSomething() {
    /* ... */
    System.out.print("A log message!");
}
```

Compliance

```
private void doSomething() {
    /* ... */
    LOGGER.log("A log message!");
}
```

Proposed Solution: Change output standards to *Loggers*

Conformity: CERT, ERR02-J. - Prevent exceptions while logging data

Competence: Techniques and Tools

R3 – Throwable.printStackTrace(...) should not be called:

Definition: The use of this feature prints the execution stack of some flow.

Problem: by default, this flow can expose sensitive information.

Related Concepts: Exceptions and Throwable, Loggers, StackTrace

Code Samples:

Non-compliance

```
private void doSomething() {
    try {
       /* ... */
    }  catch(Exception e) {
        e.printStackTrace();
    }
}
```

Compliance

```
private void doSomething() {
    try {
       /* ... */
    }  catch(Exception e) {
        LOGGER.log("context", e);
    }
}
```

Proposed Solution: it is recommended to use Loggers to define error messages.

Conformity: OWASP Top 10 2017 Category A3 - Sensitive Data Exposure, MITRE, CWE-489 - Leftover Debug Code

Competence: Techniques and Tools

Essentially, **Linked Rules List** is the core artifact, which is obtained through the evaluation of inputted (**SonarQube Violation Database**, **SonarQube Violated Rules List**, **SonarQube Conformity Rules List**, and **Violation Related Skills List**) and outputted artifacts (**Rules' Related Concepts List**). **Linked Rules List** is also used to provide support material and to generate the **Proposed Guidelines**.

Each Linked Rule consist of a set of attributes, each one directly related to presented artifacts. The attributes and a brief explanation are listed in the following:

- Definition* – a simple definition of the rule;
- Problem* – what is the problem caused by the non-compliance code;
- Related Concepts* – which concepts of the discipline is related (i.e., **Rules' Related Concepts List**);
- Code Samples – containing an example of a bad (non-compliance) and good (compliance) snippet of SC;
- Proposed Solution* – according to compliance code and conventions suggestions;
- Conformity – according to **SonarQube Conformity Rules List**;
- Competence – according to **Violation Related Skills List**;

* Definition, Problem, Related Concepts and Proposed Solution are strictly dependent on the related discipline, being evaluated by specialists in the field.

*3) **Proposed Guidelines**:* This work aims to support the teaching of programming courses, providing quality guidelines that can be used as a subsidy in conducting the teaching of programming and improving the didactic material of the discipline.

Table II and Table III are examples of proposed guidelines considering the rules demonstrated in IV-B2 (R1, R2 and R3). The set of violations selected refers to all students' submissions of all classes present on the dataset. Figure 6 represents the map of the proposed guidelines for the three showed rules, where the size of the block represents the number of violations for each rule.

TABLE II
VIOLATIONS

ID	Rule	#	%
2	Standard outputs should not be used directly to log anything	1158	65%
1	Resources should be closed	546	31%
3	Throwable.printStackTrace(...) should not be called	74	04%

TABLE III
GUIDELINES

Related Concept	R1	R2	R3	#	%
Data Input/Output	546	1158		*1704*	*96%*
Loggers		1158	74	*1232*	*69%*
StackTrace		1158	74	*1232*	*69%*
Exceptions and Throwable	546		74	*620*	*35%*
Garbage Colletion	546			*546*	*31%*
Execution Flow	546			*546*	*31%*
Constructors and Destructors	546			*546*	*31%*

V. Case Study

A case study was conducted to validate the proposed approach. We analyzed the guidelines generated for the works submitted by the summer class of 2019 and a small comparison between the results.

The analyzes were validated by a professor responsible for the discipline in a qualitative way. For this study, Work 1 (W1) and Work 2 (W2), produced in the middle and end of the 2019 class, were evaluated.

A. Violated Rules

Due to space limitation, only the 5 rules with the highest number of violations are listed both in Table IV and Table V.

As we can see, there are variations in the rules with the highest number of violations for the same group of students as the discipline evolves.

By analyzing both Table IV and Table V, it is not evident in which concepts are taught by the teacher students are facing difficulties. It becomes evident that one of the main factors that motivated us to carry out this work is below.

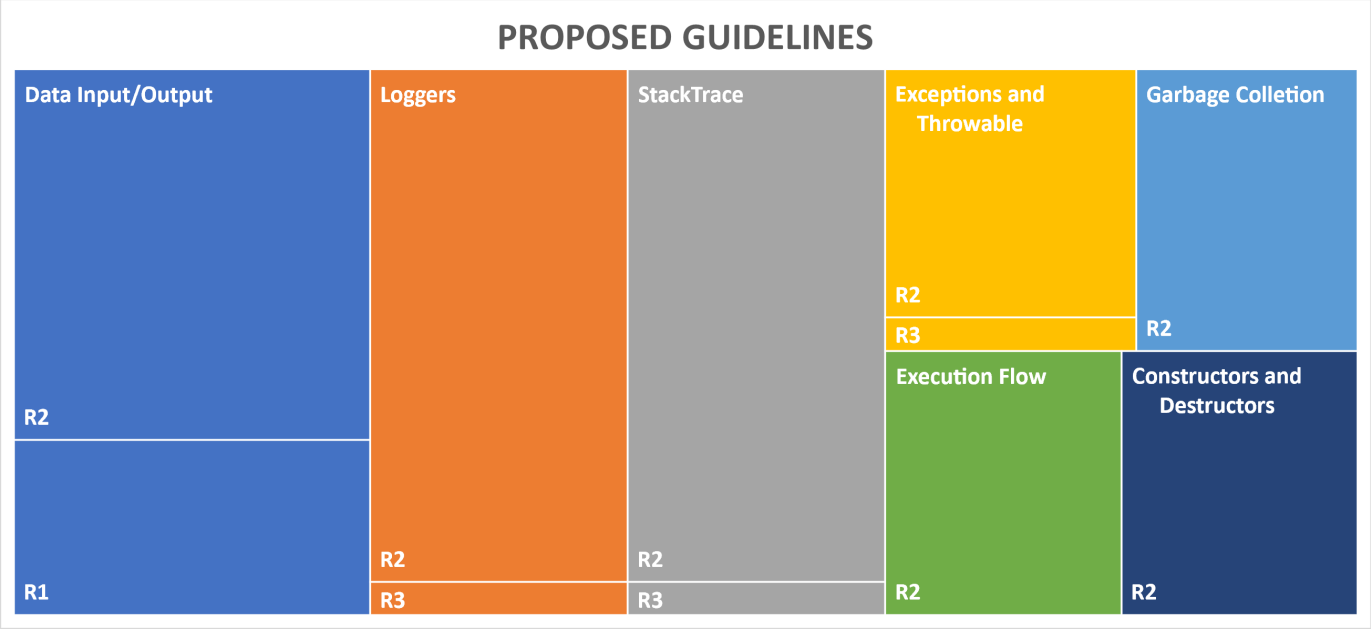

Fig. 6. Example of Proposed Guidelines for Rules Linked (IV-B2)

TABLE IV
CLASS OF 2019 – WORK 1 – TOP 5 RULES VIOLATED

Violated Rule	#	%
Standard outputs should not be used directly to log anything	286	15,91%
Strings should not be concatenated using '+' in a loop	161	8,95%
Private fields only used as local variables in methods should become local variables	149	8,29%
Multiple variables should not be declared on the same line	131	7,29%
Anonymous inner classes containing only one method should become lambdas	108	6,01%

TABLE V
CLASS OF 2019 – WORK 2 – TOP 5 RULES VIOLATED

Violated Rule	#	%
Private fields only used as local variables in methods should become local variables	687	19,22%
Anonymous inner classes containing only one method should become lambdas	412	11,52%
Unused method parameters should be removed	398	11,13%
Catches should be combined	205	5,73%
Array designators "[]" should be on the type, not the variable	198	5,54%

B. Proposed Guidelines

In order to facilitate the analyses, both Table VI and Table VII show only the five Related Concepts with most occurrence. The following results take into account all Linked Rules related to W1 and W2, respectively, according to Section IV.

By analyzing and comparing Table VI and Table VII it is evident that in the first half of the course (W1), students had less skill in using the language (Java). During the course, there was an understanding of concepts such as *Data I/O*, handling *Exceptions*.

One may see in Figure 7 a hierarchical visualization created for the lists of generated guidelines, in which the block size is associated with the number of issues and the color of the blocks with the number of rules violated.

TABLE VI
CLASS OF 2019 – WORK 1 – TOP 5 RELATED CONCEPTS

Related Concept	Rules	Issues	%
StackTrace	4	393	7.63%
Loggers	3	371	7.20%
Attributes and Variables	6	342	6.64%
Data I/O	4	338	6.56%
Attribution	4	329	6.39%

TABLE VII
CLASS OF 2019 – WORK 2 – TOP 5 PROPOSED GUIDELINES

Related Concept	Rules	Issues	%
Attributes and Variables	5	1209	11.94%
Classes	3	1180	11.65%
Instruction Blocks	2	892	8.81%
Scope	5	778	7.68%
Garbage Collection	2	725	7.16%

C. Validation

We can observe in Table IV that only the rule '*Private fields only used as local variables in methods should become local variables*' was repeated, with a considerable increase from W1 to W2.

According to the responsible teacher: "One of the reasons that justify this increase is the considerable increase in the complexity from the W1 to W2, associated with a greater understanding of concepts like *Encapsulation* and *Scope of*

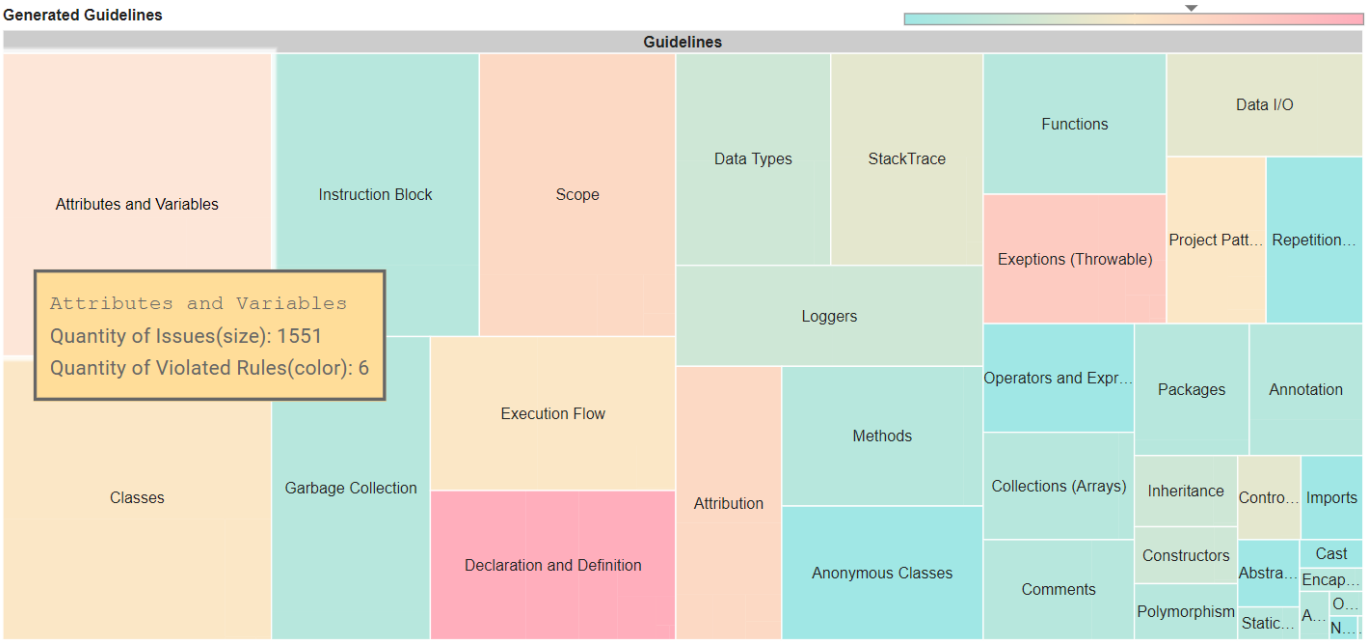

Fig. 7. Proposed Guidelines for Rules Linked, considering selection of W1 and W2 for 2019 classes.

Variables. The student who does not master these concepts well will likely do this violation several times."

In addition to the decrease in the occurrence of the rule '*Standard outputs should not be used directly to log anything*' on Table IV, indicates that the student has started to adopt other solutions instead of *System.out.print* to debug the running code.

Still, according to the professor: "The analysis based on the guidelines provides a considerable gain of time, in addition to an overview focused on the reality of the students and discipline."

VI. Final Remarks

This project defines a proposal to improve the programming teaching process, providing guidelines to the teacher to conduct programming based on students' violations. This approach makes it possible to direct the content taught by the teacher to the difficulties encountered by the students.

This project is based on a need identified by the author's research group, facing difficulty assessing the quality of what students produce, both by the teacher and the students themselves. Having empirical knowledge of the difficulties reported by colleagues in entering the labor market, it was decided to identify whether and how the situations are related to proposing improvements to the teaching-learning process, benefiting teachers and students.

For that, it was necessary to conduct a study based on the following perspectives: 1) Market expectations to understand the needs of information technology companies and what they demand from IT professionals. 2) Identifying which concepts are fundamental for software development and how these concepts are used both in industry and academic perspectives. 3) Understand the gap, why this gap exists, and what has been done to reduce it.

A. Contributions

The project's contributions are the definition of an approach to using the source code as a subsidy for quality teaching and implementing a tool that supports the defined approach, providing the teacher with a set of guidelines to support the teaching-learning process.

The Repositories of Issues (IV-A1), Related Concepts (IV-B1), Conformities (IV-A3), Competences (IV-A4) and Linked Rules (IV-B2) developed to support the methodology can be used together or separately as knowledge bases, making it possible to use them for research, study and tool support purposes.

All artifacts produced to support the proposed approach are open source, available at an open access repository, allowing the project to be adopted by the academic community.

B. Further Works

The proposal for an intervention to validate the New Module in the classroom allows participating students to benefit from the research, develop their skills, and, consequently, improve their codes' internal quality.

Acknowledgment

We want to thank the students from Laboratory of Software Engineering and Applications (LaPESA). Also, we are grateful to the Department of Mathematics and Computer Science (DMC) at Faculty of Science and Technology (FCT) by the resources granted to this research.

REFERENCES

[1] R. C. Martin, Ed., *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2009.

[2] J. Bloch, *Effective Java*, 3rd ed. Boston: Addison-Wesley Professional, 2018.

[3] R. E. Garcia, R. C. M. Correia, C. Olivete, A. C. Brandi, and J. M. Prates, "Teaching and learning software project management: A hands-on approach," in *2015 IEEE Frontiers in Education Conference (FIE)*. Ieee, 2015, pp. 1–7.

[4] L. P. Scatalon, E. F. Barbosa, and R. E. Garcia, "Challenges to integrate software testing into introductory programming courses," in *2017 IEEE Frontiers in Education Conference (FIE)*. Ieee, 2017, pp. 1–9.

[5] L. P. Scatalon, J. C. Carver, R. E. Garcia, and E. F. Barbosa, "Software testing in introductory programming courses: A systematic mapping study," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 421–427.

[6] de Andrade Gomes, Pedro Henrique and Garcia, Rogério Eduardo and Spadon, Gabriel and Eler, Danilo Medeiros and Olivete, Celso and Correia, Ronaldo Celso Messias, "Teaching software quality via source code inspection tool," in *2017 IEEE Frontiers in Education Conference (FIE)*. Ieee, 2017, pp. 1–8.

[7] de Araújo, Darlan Murilo Nakamura and Eler, Danilo Medeiros and Garcia, Rogério Eduardo, "Teacher Mate: A Support Tool for Teaching Code Quality," in *17th International Conference on Information Technology–New Generations (ITNG 2020)*. Springer, 2020, pp. 407–413.

[8] L. W. Dietz, J. Manner, S. Harrer, and J. Lenhard, "Teaching clean code," in *Proceedings of the 1st Workshop on Innovative Software Engineering Education*, 2018.

[9] Fowler, M., "Is High Quality Software Worth the Cost?" 2019. [Online]. Available: https://martinfowler.com/articles/is-quality-worth-cost.html

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

[11] M. Sami, M. Malek, U. Bondi, and F. Regazzoni, "Embedded systems education: job market expectations," *ACM SIGBED Review*, vol. 14, no. 1, pp. 22–28, 2017.

[12] J. Bailey and R. B. Mitchell, "Industry perceptions of the competencies needed by computer programmers: technical, business, and soft skills," *Journal of Computer Information Systems*, vol. 47, no. 2, pp. 28–33, 2006.

[13] G. Matturro, "Soft skills in software engineering: A study of its demand by software companies in uruguay," in *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*. IEEE, 2013, pp. 133–136.

[14] C. J. Marcarelli and L. J. Carter, "Work in progress-bridging the technology gap: an analysis of industry needs and student skills," in *2009 39th IEEE Frontiers in Education Conference*. IEEE, 2009, pp. 1–2.

[15] Rocha, João and Olivete, Celso and de Andrade Gomes, Pedro Henrique and Garcia, Rogério E and Correia, Ronaldo CM and de Souza, Gabriel Spadon and Eler, Danilo M, "Internet-based education: a new milestone for formal language and automata courses," in *Information Technology-New Generations*. Springer, 2018, pp. 195–200.

[16] S. Krusche and A. Seitz, "Artemis: An automatic assessment management system for interactive learning," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 2018, pp. 284–289.

[17] "SonarQube Documentation | SonarQube Docs." [Online]. Available: https://docs.sonarqube.org/latest/

[18] S. Committee, *ACM Curricula Recommendations for Computer Science 2020 – Paradigms for Global Computing Education*. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf

[19] A. F. Zorzo, D. Nunes, E. S. Matos, I. Steinmacher, J. C. Leite, R. Araujo, R. C. M. Correia, and S. Martins, *Referenciais de Formação para os Cursos de Graduação em Computação 2017*. São Paulo: Sociedade Brasileira de Computação (SBC), 2017. [Online]. Available: http://www.sbc.org.br/documentos-da-sbc/send/131-curriculos-de-referencia/1165-referenciais-de-formacao-para-cursos-de-graduacao-em-computacao-outubro-2017