I.J. Modern Education and Computer Science, **2018, 10, 31-39**
Published Online October 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2018.10.04

Framework for Software Code Reviews and Inspections in a Classroom Environment

Fernando Almeida
Faculty of Engineering of Oporto University, INESC TEC, Porto, Portugal
Email: almd@fe.up.pt

Received: 17 August 2018; Accepted: 05 September 2018; Published: 08 October 2018

***Abstract*—**Code reviews and inspections have the purpose to ensure that the code has sufficient quality to be released. It is generally seen as an economical way of finding errors, increase team productivity and sharing technical and product knowledge among team members. This approach is traditionally adopted in software development companies, but their practices may be useful in other contexts, such as in the process of learning software engineering. In this sense, this study proposes an innovative framework for conducting code reviews in a Computer Science course. The proposed framework can be applied in any object-oriented program language, and it is sufficiently concise to be applied in the classroom, namely in a 90-minute session in which all students are invited to collaborate in this process. The findings suggest that code reviews in an academic context can help students to strategically reflect about the performed work, enhance their soft skills, and increase their ability to work in groups. On the other hand, as the main challenges, the findings reveal that students typically don't have previous experience in performing inspections and it can become difficult to perform a complete inspection in a classroom session.

***Index Terms*—**Code Review; Education; Learning; Software Engineering; Software Development; Software Quality.

I. Introduction

Software engineering is an engineering field that is concerned with all aspects of software process, from the initial stage of the system specification to customer maintenance [1]. The software engineering field is necessarily multidisciplinary and requires the knowledge and application of various knowledge domains, such as, design, workflow, project management, testing, databases, quality control, requirements, architecture, programming, cost estimation, and law and ethics.

The software development process incorporates a set of methods, tools and processes to analyze, design and develop software with quality and within the estimated time frames and costs. Factors such as effort, productivity, time, cost of development and quality are negatively affected when software artifacts are produced due to the work required to correct these defects. In [2,3] it is also known that the cost of labor for defect correction increases as the development process progresses. In this way, initiatives to correct errors and anomalies must be carried out as soon as possible. An approach that has proven to be efficient and cost-effective in finding defects, reducing effort, and improving product quality is the review and inspection of artifacts produced throughout the software development process [4].

Considering the need to bring academia and industry closer together through teaching and learning of software engineering, this study seeks to encourage the development of code review practices among students attending software engineering courses in higher education. To this end, an innovative multi-dimensional framework is proposed, which can be used by students to analyze the maturity and quality of their software development practices in a practical software engineering project. The use of code reviews in the classroom and throughout the student training process will become important in the acquisition of good programming practices, in the identification of the importance that the software quality assumes in the software development process, and in the their subsequent insertion in the labor market.

This study has as the main research question the exploration of the main benefits, challenges and difficulties brought by the adoption of this framework for code reviews. The manuscript is organized as follows: Initially, a review of the literature on the process of conducting code reviews, its impact at the enterprise level and its application in the classroom is carried out. Then, the approach followed for defining a guide for the formulation of code reviews is presented. Subsequently, the working methodology is presented and then the main results of the process are presented and discussed. Finally, the conclusions of this study are drawn.

II. Literature Review

The Institute of Electrical and Electronic Engineers (IEEE) establishes five types of reviews [5]: (i) management review, which is as a systematic evaluation process of acquisition, development, operation and maintenance processes performed by managers; (ii) technical review, which is a peer review approach in which technical qualified personnel examines the

software to identify discrepancies from specifications and standards; (iii) inspections, which involve a rigorous process to detect and correct defects; (iv) walk-through, which incorporates joint review effort to improve product quality in software development work; and (v) audits, which is an internal or external review of a software program to check its quality, progress or adherence to plans, standards and regulations.

There are several models and formal techniques for the execution of an inspection, such as Fagan methodology, Glib methodology, phase inspection, scenario based inspection method and Defect Management Oriented Inspection (DEMAO). Traditional inspection process defined by Fagan in 1976 is still the most used in industry, which is composed of five iterative and sequential steps [6]: (i) overview; (ii) preparation; (iii) inspection meeting; (iv) rework; and (v) follow up. In [7] it is stated that the evolution of these techniques has the following objectives: (i) improve quality; (ii) improve efficiency; (iii) increase reliability; (iv) reduce effort; (v) reduce time of meetings; (vi) increase the defects detection; and (vii) reduce complexity.

One of the most studied ways in the literature to increase the quality of software is through the use of software inspection. This approach is defined as a particular type of review that can be applied to all software artifacts and has a rigorous and well defined defect detection process [8]. Additionally, empirical research in this field clearly evidences that inspections generally benefit software development process and quality assurance [9]. Consequently, several authors have suggested using different approaches to software inspection to increase software quality. In [10] it is suggested the use of formal inspection, which may be applied to any product or partial product of the software development process. In [11] it is proposed a system dynamics model for simulation of the software inspection process. Finally, in [12] it is proposed the adoption of management and technical review techniques, which can drastically reduce the time and costs required for testing, debugging and reworking.

In [13] it is pointed out nine benefits offered by software inspection practices: (i) minimize the chances of defects reported by users; (ii) customer satisfaction is increased; (iii) amount of productivity is also increased; (iv) increase in-time delivery of software projects; (v) help in meeting the committed schedules efficiently; (vi) increase the experience and speed up the cross-training of team member on new products; (vii) improve the development process model; (viii) provide team building environment; and (ix) can potentially eliminate the need of unit testing of code, in some cases. This vision is confirmed in [14] by stating that inspections play a valuable role in training new employees. He advocates that software inspections are useful for educating new employees on the practices and processes employed in the organization.

Inspection sessions must be pre-scheduled and planned. In [13] it is suggested the existence of five roles: (i) the moderator, who is the leader of inspection activity; (ii) the author, who is responsible for the creation and maintenance of the work product that is to be inspected; (iii) the reader; who reads the work product to the team; (iv) the recorder, who records the defects and issues that were founded during the inspection activity; and (v) the inspector, who tries to find errors in the work product. In these roles emerge the critical function deployed by an inspector that must be a skilled and experience individual, typically a senior programmer.

It is also relevant to classify the type of errors, since there may be multiple dimensions with different levels of criticality. In [15] it is defined eight dimensions: (i) omission; (ii) ambiguous; (iii) inconsistent; (iv) superfluous; (v) incorrect; (vi) not-conforming to standards; (vii) not-implementable; and (viii) risk-prone. It is important to recognize that the same error can be cataloged in different ways, according to the specificity of each programming language and the way the program is built. For instance, a global variable later declared to be local in a given method can be considered an ambiguity or an inconsistency, in the light of its use. It is also important to adopt measurements to monitor and analyze the success of an inspection. For that, in [16] it is proposed nine key metrics: (i) total KLOC inspected; (ii) average LOC inspected; (iii) average preparation time; (iv) average inspection rate; (v) average effort per KLOC; (vi) average effort fault detected; (vii) average faults detected per KLOC; (viii) percentage of re-inspections; and (ix) defect-removal efficiency.

Often in software field the terms code review and software inspections are used undifferentiated. In fact, the two terms share the same objectives, but it is important to clarify that according to [17] they vary in amount of planning required, the amount of formality, number of people and number of roles. It is correct to consider that more heavyweight approaches like software inspections tend to be more effective because they have potential to detect more software errors. However, this approach tends to be less efficient due to the high consumption of time and resources and therefore its practical use is often impracticable [18].

There are several types of code reviews with different scope. The evolution of new software development processes, such as the emergence of agile methodologies and lean programming, has also led to the emergence of more dynamic and interactive code review models. In [19] it is considered the existence of four code review types: (i) over-the-shoulder, one developer is responsible to look to the code developed by other colleague as the latter walks through the code; (ii) email pass-around, source code management systems emails code to reviewers automatically; (iii) pair programming, two programmers develop code together at the same workstation; and (iv) tool-assisted code review, authors and reviewers use specialized tools to assist the peer code review. At scope level, we may find a wide range of models [19]: (i) goal review; (ii) API/design review; (iii) maintainability review; (iv) security review; (v) integration review; (vi) testing review; and (vii) license review.

Maintaining quality in software development is a challenge faced by many companies. Several strategies can be used to check software quality, such as testing, software reviews, patterns and software metrics [20]. Furthermore, the process of conducting a code review and its coverage are elements that influence the software quality. This confirmation is given by [21] that establish a taxonomy of five factors in a code review that influence the obtained results and the quality of the software: (i) product (e.g., size and complexity); (ii) process (e.g., prior defects, churn and change entropy); (iii) human factors (e.g., total authors, minor/major authors and authors ownership); (iv) coverage (e.g., proportion of reviewed changes and proportion of reviewed churns); and (v) participation (e.g., proportion of self-approved, proportion of hastily reviewed changes and proportion of changes without discussion). On the same direction, in [22] it is stated that large and more complex components are more likely to be defect-prone. Additionally, in [23] it is referred that components that have undergone a lot of change are likely defect-prone.

The effectiveness of code reviews is also investigated in the literature. In [24] it is found that number of lines of code and complexity of a program affect the effectiveness and efficiency of code review. On the other hand, in [25] it is stated that number of involved teams, participants and locations generally improve reviewer contributions, but with a severe penalty to the duration. In this sense, several authors appear to propose strategies to increase the effectiveness of a code review. Additionally, code standards and informative comments are useful to ensure consistent flow of information among teams over the project lifecycle [26]. Furthermore, the adoption of code review tools that can help identify some potential issues via inspections [27].

In [28] it is summarized the top five benefits offered by code reviews: (i) finding defects; (ii) code improvement; (iii) look for alternative solutions; (iv) knowledge transfer; and (iv) team awareness and transparency. Code reviews help code to become simpler, clearer, and better understandable [29]. Additionally, it contributes to improve the feeling of collective code ownership. In [30] it is explored the role of code reviews at Microsoft through the use of an qualitative empirical study. They concluded that code reviews can also be very useful for new team members to learn the project design, constraints, available tools and application programming interfaces (APIs).

Despite unequivocal advantages associated with code review processes that are generally identical to those identified in software inspections, resistance to the implementation of code reviews is still experienced in many companies. Two primary reasons can be identified [31]: programmer egos that tend to put obstacles in the code to be revised by other programmers, and the hassle of packaging source code for review and scheduling review meetings. In [32] it is suggested the existence of high costs associated with inspections as a technology. Additionally, it is important to have in mind that code reviews catch only about half of the defects [26]. Therefore, additional verification and validation (V&V) tools and techniques are required to ensure trustworthy code.

Studies reporting the use of code review and inspection techniques in the classroom are very limited. One of these studies is written by [33], in which three classroom exercises were created to detect errors in object-oriented systems. This study concludes that traditional reading techniques are not appropriate in the process of inspecting the code in large object-oriented (OO) systems, because this development paradigm can lead to delocalization problems. These issues emerge due to the need to have a dynamic view of the system and a global perception of the software. In this sense, other techniques are suggested like the adoption of a use-case driven strategy and creation of personalized checklists. Study [34] invites students to participate in their own learning process as part of a community of learners through the adoption of code reviews. In [35] it is reported that the use of code reviews by software engineering students improve their own self-evaluation and confidence in their abilities. Finally, in [36] it is included near-peer mentor preparation and code reviews to expand capacity and promoting students' inclusion in introductory computer science courses.

III. Approach

Several authors have proposed frameworks to conduct a code review process. Some of these proposals are specific to a given programming language (e.g., Java, Python, C# / C++ and Ruby on Rails), while others take a generic perspective and approach cross-points to all object-oriented programming languages. There are frameworks proposed informally by programmers, project managers, consultants and software engineering companies. In this study, all these proposals were not considered, but only proposals published in books, chapters of books, journals and indexed international/national conferences. Additionally, code review frameworks adopted in the context of higher education institutions, particularly in computer science or similar courses were considered.

In [17] it is proposed a generic code review composed of six dimensions: (i) structure; (ii) documentation; (iii) variables; (iv) arithmetic operations; (v) loops and branches; and (vi) defensive programming. This latter dimension is clearly the most innovative and basically comprises a set of issues relating to error detection, memory allocation and performance. In total, 35 questions are considered.

In [31] it is proposed a code review framework composed of five dimensions: (i) documentation; (ii) testing; (iii) error handling; (iv) thread safety; and (v) performance. In total they propose a checklist composed of 25 items. They also advocate that longer checklists tend to be less effective and, therefore, they propose keeping it down to the 20-25 most critical items [31].

In [37] it is suggested seven dimensions: (i) feature; (ii)

background; (iii) scenarios; (iv) tags; (v) general code; (vi) steps; and (vii) exceptions. In total In total, 36 questions are presented divided by seven dimensions in a relatively asymmetric way since, for example, the "feature" dimension has 8 items, while the "exceptions" dimension has 2 items. Some of these items are also relatively ambiguous and redundant, since if the code follows the coding standards, then it must necessarily be well-structured and consistent in style and formatting, since this is one of the rules that must be explicitly defined in a coding standard.

In [38] it is proposed a specific code review for embedded systems composed of seven dimensions: (i) function; (ii) style; (iii) architecture; (iv) exception handling; (v) timing; (vi) validation and testing; and (vii) hardware. Additionally, the authors suggest that each dimension should be given to a specific reviewer and the review should include 100-400 lines of code per 1-2 hour review session. In total, this model proposes 62 questions, which turn it very complete and exhaustive. It is relevant to highlight that this model is adopted in the College of Engineering of Carnegie Mellon University.

In [39] it is presented a specific code review guideline that is used in the context of Java software programming classes at the Department of Computer Science and Software Engineering of Cal Poly College of Engineering. The framework has ten dimensions: (i) specification/design; (ii) initialization and declarations; (iii) method calls; (iv) arrays; (v) object comparison; (vi) output format; (vii) computation, comparisons and assignments; (viii) exceptions; (ix) flow of control; and (x) files. In total, 45 questions are proposed.

In [40] it is used in the context of a software engineering course at Paul G. Allen School of Computer Science & Engineering of University of Washington a code review framework composed of five dimensions: (i) coding standards; (ii) comments; (iii) logic; (iv) error handling; and (v) coding decisions. This model also proposes the existence of a section for review notes in which the reviewer must expose the founded problems and decisions made. In total, 35 questions were considered.

A comparative analysis of these approaches is performed in Table 1. For this purpose some of these dimensions were aggregated, since they generally approach the same items. The following acronyms are used: "-" means that this dimension is not found in a given study; "Y" the dimension is explicitly mentioned in a given study; whereas "P" means that the dimension is only implicitly considered. The adopted terminology in the organization of dimensions is distinct from several authors. However, issues that are addressed in the review process are similar among them. An example of this situation is the concept of "defense programming" introduced by [17] and whose practices are adopted in the model proposed by [31,38, 40]. In fact, the last author addresses essentially the same content, but uses "code decisions" terminology. For instance, in this section, they propose to analyze whether redundancy is minimized; defensive copies are made when needed, no unnecessary new objects are created, etc. For his part, in [37] it is proposed the "background" and "scenarios" sections that basically correspond to "structure" and "testing" of other frameworks. In general terms, "structure", "variables" and "error handling" are the most commonly found dimensions in a code review guideline.

Table 1. Classification and comparative analysis of code review dimensions

Dimension	**Ref [17]**	**Ref [31]**	**Ref [37]**	**Ref [38]**	**Ref [39]**	**Ref [40]**
Structure	Y	P	Y	P	Y	P
Documentation	Y	Y	-	-	-	Y
Variables	Y	P	P	P	Y	P
Arithmetic operations	Y	P	-	P	Y	P
Loops and branches	Y	Y	-	P	Y	P
Defensive programming	Y	P	-	P	-	Y
Testing	-	Y	P	Y	-	P
Error handling	P	Y	Y	Y	Y	Y
Performance	P	Y	-	Y	-	-
Feature	P	-	Y	Y	Y	-
Output format	P	-	-	-	Y	-
Files	Y	-	-	-	Y	Y
Coding standards	Y	-	Y	-	-	Y
Hardware	-	-	-	Y	-	-

Before defining the structure of code review that we intend to adopt in the context of the computer science course, some restrictions were defined considering the structure of the course, students' profile and classes' organization. Thus, three premises must be accomplished: (i) it must be used in the context of programming classes for desktop, mobile and web environment; (ii) it must be appropriate to the profile of an undergraduate student; (iii) it should promote the increase of the quality of software developed in the academic context; and (iv) it must have a reduced size, 20 - 25 most critical item as suggested by [31], so that it is not an inhibiting element of learning and can be carried out in a 90-minutes session.

Table 2 presents the adopted framework that has been utilized during the last three academic years. The framework has 25 questions organized in seven dimensions. In the "feature" dimension, we intend to verify the code's compliance with the functional requirements and architecture of the application. This section is considered absolutely essential in the process of teaching software engineering, since it is crucial that students realize the importance of the requirements capture process and its correct mapping with the code. In the "structure dimension" the importance of coding standards is emphasized, which is also a document that students should prepare before starting their group work. In the third dimension, we organized together the variables, operations, loops and branches elements, since they are related to technical decisions of the code implementation. "Error handling" is also another dimension contemplated in framework, which becomes

fundamental in the process of increasing the quality of the code. Subsequently, appears the "documentation" section, which is a key element in the code maintenance process. "Testing" is another element considered fundamental, namely the existence of unit tests on the developed code. Finally, "performance" is also another dimension addressed by several studies. This last dimension tries to instill in the students at an early age the need to have the code properly developed considering demanding scenarios, in which the capacity of the device is limited or the system must offer a real response time.

IV. Methodology

The methodology is divided into four phases as presented in Fig. 1. In the preliminary stage, a review of the literature on code reviews and inspections is carried out, in which it is intended to explore the importance of these techniques in the software engineering field. Additionally, this phase intends to perform a comparative analysis of proposals for structuring code reviews according to several dimensions. Then, in the conceptual phase, the adopted framework for academic code reviews is presented. Subsequently, in the exploitation stage, an empirical study of the application of the previously formulated framework proposal is performed. Finally, in the fieldwork stage, the benefits, challenges and limitations of this approach are explored.

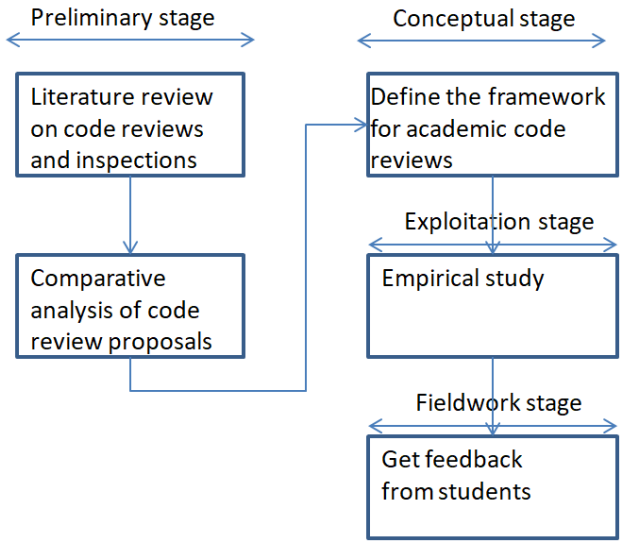

Fig.1. Research methodology

The empirical study occurred during the last three academic years, since 2015/16 to 2017/18. It took place in a University environment using 3rd year Computer Science students who had during their academic journey two years of programming experience in Java, C# and PHP. The students are distributed by several teams, in which each team there is a project manager, two software analysts and between 3 and 4 programmers. The waterfall methodology is used in the software development process. In the process of conducting the inspection the code is presented by the student who assumes the role of project manager, in which follows the guide of table 2. The inspection is done in the classroom in a session of 90 minutes. The dimension of the teams and objectives of each project is described in table 3. The technologies adopted in the development of the project were the same despite the distinct objectives of each project in each academic year.

Table 2. Adopted framework for academic code reviews

Feature
F1. Does the code completely and correctly implement the functional requirements?
F2. Does the code fit the architecture's design?
F3. Is there any excess functionality in the code but not described in the specification?
Structure
S1. Does the code conform to any pertinent coding standards?
S2. Are any modules excessively complex and should be restructured or split into multiple routines?
S3. Can any code be replaced by calls to external reusable components or library functions?
Variables, Operations, Loops and Branches
VOLB1. Do all assigned variables have proper type consistency or casting?
VOLB2. Are there any redundant or unused variables?
VOLB3. Does the code systematically prevent rounding errors?
VOLB4. Are divisors tested for zero or noise?
VOLB5. Are all loops, branches, and logic constructs complete, correct, and properly nested?
VOLB6. Are indexes or subscripts properly initialized, just prior to the loop?
Error handling
EH1. Are input parameters checked for proper values?
EH2. Are error messages understandable and complete?
EH3. Are all relevant exceptions caught?
Documentation
D1. Is the code clearly and adequately documented with an easy-to-maintain commenting style?
D2. Are complex algorithms and routines properly explained and justified?
D3. Are all comments consistent with the code?
Testing
T1. Do unit tests have 100% branch coverage?
T2. Are all interfaces tested, including all exceptions?
T3. Does the code provide convenient ways to inject faulty conditions for testing?
Performance
P1. Is every memory allocation deallocated?
P2. Is memory usage acceptable even with large inputs?
P3. Can better data structures or more efficient algorithms be used?
P4. Has code readability been sacrificed for unnecessary optimization?

The students' opinion collection was captured after the conclusion process of each code review. In each academic year two code reviews were established: one of them after three months of the project initiation; the last code review, two weeks before the delivery of the project. This model is common for the 3 academic years. The programmers responsible for the development of the code are responsible to prepare the source code for review, which implies format and document properly the source code.

A key question in the process of performing a code review is to define what sections should be included or not in the review process. We generally adopt the recommendations suggested by [17] that best suitable sections to be included in code reviews include complex

logic code, implementation of algorithms, and code whose bad construction has a significant impact on the overall system. This study just doesn't follow the recommendation to choose code typically designed by new or inexperienced team members, because in academic context we consider that all students have similar maturity in code development. On the other side, reused code, repeated parts of code and parts of the code that, if faulty, are not expected to affect functionality.

In order to evaluate the development process of the code reviews a qualitative approach was used through the adoption of semi-structured interviews. The qualitative methodology allows an in-depth analysis of the data in order to perceive the behaviors and tendencies of a target audience. This detailed exploration approach allows a deeper understanding of the causes of a given behavior, which is one of the main advantages associated with this research method [41].

Table 4 presents the guide for the development of the interviews, which is grouped in three dimensions: (i) contextual; (ii) evaluative; and (iii) strategic. Semi-structured interviews are especially recommended for group interviews and allow a more systematic treatment of the data [42,43]. Additionally, it allows the introduction of new informal questions throughout the interview according to the feedback received by each group.

Table 3. Description of the projects and involved teams

Academic year	Students	Project goal
2015/16	2 groups of 7 students	The goal of this project is to develop a loyalty card application for a supermarket. The concession of discounts to the customer is based on the type of products purchased by them and the existence of promotional campaigns.
2016/17	2 groups of 7 students	This project provides a car-sharing solution. With this application the user can rent a vehicle during a period of time. This application helps the customer in the process of choosing the best car service and vehicle suited to his/her needs.
2017/18	3 groups of 6 students	This project has developed an application that aims to assist in the process of composing software development teams. To this end, this application helps software engineering companies to formulate a Scrum team consisting of product owner, Scrum master and Scrum team.

Table 4. Interview' structure

Dimension	Questions
Contextual	Q1. What is the state of project' development? Q2. What is the team's involvement in the project?
Evaluative	Q3. What is the result of the evaluation adopted the proposed framework?
Strategic	Q4. What are the main benefits of adopting the framework? Q5. What are the main challenges and limitations of adopting the framework?

V. Results and Discussion

A. Contextual dimension

Q1: What is the state of project' development?

A contextual aspect that was initially assessed is the state of development of the source code at the time of the code review. In each academic year two code reviews were carried out at two key moments in the project development process (i.e., one after 3 months of the project' kickoff, the other two weeks before project delivery). Despite the difference between the objectives of each project, the following common events occurred:

• After 3 months of project start-up in all academic years, there were significant delays in developing the code at this stage. Two reasons conditioned the development of the code: (i) time needed by the students to analyze the requirements, definition of the system architecture and design of the database; and (ii) students' level of knowledge in object-oriented programming was reduced despite prior academic programming experience in Java, C# and PHP. This situation occurs because the programming knowledge in object-oriented languages was essentially explored in the resolution of small exercises, without the need to build a project that requests integration of different technologies;

• At the time of the last code review, there were very heterogeneous levels of development of the source code, particularly in the 2016/17 academic year, in which one group had only implemented less than 50% of the functional requirements defined in the project' kickoff. The main reason is the difficulty experienced by students in attending various curricular units in parallel with numerous written and practical assessment tests.

Q2: What is the team's involvement in the project?

It was verified, as mentioned in the previous point, delays in the development of the projects. In fact, only 2 out of 7 projects were able to implement all the functional requirements within the deadlines defined by the school calendar (one semester). The involvement of students in each project was conditioned by the frequency in the course of students with some curricular units in arrears, which strongly conditioned the students' willingness to attend all classes. In these groups more delays in the development of the project occurred. Synchronous and asynchronous communication technologies, such as chats, forums, and social networks, helped to mitigate this issue.

B. Evaluative dimension

Q3: What is the result of the evaluation adopted the proposed framework?

The results obtained by the students' participation in the code review give us some relevant indicators that should be analyzed in each of the phases.

• Phase I (first code review after 3 months of project' kick-off) – due to delays across all groups, the response to some of the code review questions is inconclusive. In three groups it was verified that the architecture of the

application was incomplete, namely in the definition of the logical architecture and in the design of the database. Despite this, and through the code developed by each group at this stage, there was a complete correction in the consistent use of the type of variables and in the documentation of the produced code. These were the two main positive indicators collected in this first code review held in each academic year;

• Phase II (second code review realized two weeks before project delivery) – the obtained results allow us to verify that: (i) not all functional requirements were implemented in the projects. This situation was not critical, since there was a concern in the prioritization of the implementation of the requirements, and was given preference to the implementation of high and medium priority requirements; (ii) new functional requirements emerged throughout the project development phase, being reflected in the code produced, but not in the specification and modeling of the requirements; (iii) substantial part of the implemented algorithms resulted in overly complex and redundant code that could be replaced using reusable components or library functions; (iv) approximately half of the projects used explicit conversions of the data type and their majority (5 in 7 groups) used redundant variables or global variables without any valid reason; and (v) most of the groups focused their attention on code production without considering the importance of generating automatic unit tests, interface tests, robustness tests, among other kind of tests.

C. Strategic dimension

Q4: What are the main benefits of adopting the framework?

The benefits reported by students are globally common to the benefits summarized in the literature review. Among them are the benefits related to improving the final quality of the code, finding alternative solutions to the same problem and improving the code readability. Several students pointed out that comments in the code were only included motivated by the existence of code reviews. Additionally, code review helps students to follow coding standards compliance which helps them to maintain a consistent coding style. In the project of this last academic year was also pointed out by one of the groups that the code review helped them to find problems in little executed parts of the code, which was not covered by the specified unit tests.

Other encountered benefits appear in the context of the application of code reviews in the classroom. Students stated that it was very important to perform the code review in a 90-minute class. Its realization in a classroom was also considered very important because it allowed the presence of all students. These sessions allowed the students to look critically at the work produced, something that is typically not explored in the classroom context. It was stressed by the students the importance of the existence of moments that allow an informal reflection and evaluation of the work produced during the semester, without the need for a quantitative evaluation.

Increasing cohesion within the working group was another relevant benefit. All teams noted difficulties experienced by some students in dealing with the pressure and manage their soft skills. At critical moments of the project, particularly those close to milestones, some fragmentation within these groups was felt. Code review discussions helped to save team members from isolation and bring them closer to each other. Additionally, it was pointed out transversally that the code review sessions were fun and helped the students to feel more involved with the project. Finally, all groups indicated that mistakes made in code development served as individual and collective learning so that the same mistakes were not made in future projects, particularly in the context of other curricular units offered by the Computer Science course.

Q5: What are the main challenges and limitations of adopting the framework?

The challenges and limitations found by [31,32] related to the existence of egos, problems of packing source code for review, and the existence of high costs were not verified in this study. All students that assumed the role of programmers in their teams expressed total openness to let the code being analyzed by their colleagues. They looked at code reviews as a helpful process for code improvement. In addition, no difficulties were experienced in preparing the code for review because Apache Subversion was used as a version control system.

However, other challenges and difficulties were experienced in the operationalization of classroom inspections. At the time of the 1st code review, it was verified that most of the groups had developed little code, so that the result obtained with this first code review was generically common among all groups. Most of the groups mainly inspected the user authentication process and introduced code improvements to ensure password encryption in the database and exception handling in the application. The scenario changed considerably when the 2nd code review was performed, in which the amount of code developed was already significantly larger. However, due to this situation, it was not possible to review the entire code in a 90-minute session. As solution, in the process of conduction the review, preference was given to high priority requirements. Another difficulty was the students' lack of experience in performing software inspections. It was the first time they came into contact with this reality and, therefore, doubts emerged that were clarified in the classroom. This first experience faced by students was emphasized by them as being very positive and that surely will help them in the future to have a better performance in future code reviews in academic and business context.

VI. Conclusions

Code review is a software review practice that has been used by several software development companies. Generally, it consists of some or all members of a software development team reviewing a colleague's code before integrating it into the production version of the code. Several benefits have been generically associated with this practice as an economical way of finding errors, improving code quality, increasing productivity and sharing product knowledge. However, this technique has been little explored in the classroom and, therefore, this study proposes a framework for conducting code reviews in a Computer Science course.

The results suggest that the benefits of using code reviews in a classroom environment are generically similar to those found in project management teams in business environments. In addition to these benefits, it is important to recognize the importance that these code reviews can have in the students' reflection on the developed work, in the development of soft skills, in teamwork and in individual and collective perception that these mistakes can help them throughout the course. On the other hand, as the main challenges, we have the difficulty of its operationalization in the classroom in due time, especially when the volume of code produced is high, and the students' lack of experience in performing code inspections.

The practical implications of this study are substantial for Computer Science courses since we advocate that code reviews should be encouraged in the classroom. In software projects of medium-high complexity, in which the development of software takes the entire semester, it is important to have code reviews that assist the code development process and contribute to the cohesion of working groups. Given the obtained results, it can be verified that the number of code reviews performed was insufficient, being recommended the existence of code reviews more uniformly distributed in the last months of the project. It may make more sense to have code reviews when the implementation of functional requirements is completed and not in fixed positions in time.

As future work we intend to explore the inclusion of code reviews in agile development environments. We also intend to explore the use of automated code review tools and formulate a model in which it is possible to jointly use the combination of manual and automated efforts.

References

[1] I. Sommerville, *Software Engineering*. Pearson Education, 2015.

[2] M. Zhivich and R. Cunningham, "The Real Cost of Software Errors", *Secure Systems*, vol. March/April, pp. 87-90, 2009.

[3] L. Bergmane, J. Grabis and E. Zeiris, "A Case Study: Software Defect Root Causes", *Information Technology and Management Science*, vol. 20, pp. 54-57, 2017.

[4] J. Dooley, *Walkthroughs, Code Reviews, and Inspections*. Apress, 2011.

[5] IEEE, "IEEE Standard for Software Reviews and Audits", IEEE Standards. Retrieved 2018, May 28, from https://ieeexplore.ieee.org/document/4601584/

[6] A. Mishra and H. Shukla, "Software Inspection: An Overview", *International Journal of Advanced Computational Engineering and Networking*, vol. 1, no. 5, pp. 32-34, 2013.

[7] A. Qazi, S. Shahzadi and M. Humayun, "A Comparative Study of Software Inspection Techniques for Quality Perspective", *International Journal of Modern Education and Computer Science*, vol. 10, pp. 9-16, 2016.

[8] Y. Zhu, *Software Reading Techniques: Twenty Techniques for More Effective Software Review and Inspection*. Apress, 2016.

[9] S. Kollanus and J. Koskinen, "Survey of Software Inspection Research", *The Open Software Engineering Journal*, vol. 3, pp. 15-34, 2009.

[10] T. Devi, "Improving Quality of Software through Formal Inspection", *International Journal of Engineering Research and Application (IJERA)*, vol. 2, no. 1, pp. 552-557, 2012.

[11] J. Coelho, J. Braga and B. Ambrósio, "System dynamics model for simulation of the software inspection process", *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 5, pp. 1-8, 2013.

[12] I. Akpannah, "Optimization of Software Quality using Management and Technical Review Techniques", International *Journal of Computer Trends and Technology (IJCTI)*, vol. 17, no. 6, pp. 304-309, 2014.

[13] A. Ahad, Z. Ullah, L. Tariq and S. Niaz, "Software Inspections and Their Role in Software Quality Assurance", *American Journal of Software Engineering and Applications*, vol. 6, no. 4, pp. 105-110, 2017.

[14] G. O'Regan, *A Practical Approach to Software Quality*. Springer-Verlag, 2012.

[15] A. Alshazly, A. Elfatatry and M. Abougabal, "Detecting defects in software requirements specification", *Alexandria Engineering Journal*, vol. 53, pp. 513-527, 2014.

[16] G. Huzooree and V. Ramdoo, "Evaluation of Code Inspection on an Outsourced Software Project in Mauritius", *International Journal of Computer Applications*, vol. 113, no. 10, pp. 39-44, 2015.

[17] K. Wiegers, *Peer Reviews in Software: A Practical Guide*. Addison-Wesley Professional, 2001.

[18] L. Copeland, *A Practitioner's Guide to Software Test Design*. Artech House, 2013.

[19] A. Bhuyan, "Code Review Principles, Processes and Tools", Retrieved 2018, May 29, from https://www.scribd.com/document/291887013/Code-Review-Principles-Process-and-Tools

[20] D. Galin, *Software Quality: Concepts and Practice*. Wiley-IEEE Computer Society, 2018.

[21] S. McIntosh, Y. Kamei, B. Adams and A. Hassan, "The impact of code review coverage and code review participation on software quality: a case study of the qt, VTK, and ITK projects", *Proceedings of the 11th Working Conference on Mining Software Repositories*, Hyderabad, India, pp. 192-201, 2014.

[22] A. Koru, D. Zhang, K. Eman and H. Liu, "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules", *Transactions on Software Engineering (TSE)*, vol. 35, no. 2, pp. 293-304, 2009.

[23] N. Nagappan and T. Ball, "Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical

Case Study", *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Madrid, Spain, pp. 364-373, 2007.
[24] S. Ahmed and R. Purohit, "Evaluating Efficiency and Effectiveness of Code Reading Technique with an Emphasis on Enhancing Software Quality", *International Journal of Computer Applications*, vol. 2, pp. 32-36, 2014.
[25] E. Santos and I. Nunes, "Investigating the Effectiveness of Peer Code Review in Distributed Software Development", *Proceedings of the 31st Brazilian Symposium on Software Engineering*, Fortaleza, Brazil, pp. 84-93, 2017.
[26] S. Nelson and J. Schumann, "What makes a Code Review Trustworthy?", *Proceedings of the Thirty-Seventh Annual Hawaii Int. Conf. on System Sciences (HICSS-37)*, Hawaii, USA, pp. 1-10. 2004.
[27] T. Gee, "Ways to Make Code Reviews More Effective", Retrieved 2018, May 30, from https://www.infoq.com/articles/effective-code-reviews
[28] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review", *Proceedings of the 2013 International Conference on Software Engineering*, San Francisco, USA, pp. 712-721, 2013.
[29] J. McCrary, "An Effective Code Review Process", Retrieved 2018, May 30, from https://jakemccrary.com/blog/2014/12/09/an-effective-code-review-process/
[30] A. Bosu, M. Greiler and C. Bird, "Characteristics of useful code reviews: an empirical study at Microsoft", *Proceedings of the 12th Working Conference on Mining Software Repositories*, Florence, Italy, pp. 146-156, 2015.
[31] J. Cohen, S. Teleki and E. Brown, *Best kept secrets of peer code review*. SmartBear Software, 2013.
[32] N. Fogelström and T. Gorschek, "Test-case Driven versus Checklist-based Inspections of Software Requirements – An Experimental Evaluation", Retrieved 2018, May 29, from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.218.5538
[33] A. Dunsmore, M. Roper and M. Wood, "Practical Code Inspection Techniques for Object-Oriented Systems: An Experimental Comparison", *IEEE Software*, vol. 20, no. 4, pp. 21-29, 2003.
[34] H. Sondergaard and R. Mulder, "Collaborative learning through formative peer review: Pedagogy, programs and potential", *Computer Science Education*, vol. 22, pp. 343-467, 2012.
[35] Y. Wang, H. Li, Y. Feng, Y. Jiang and Y. Liu, "Assessment of programming language learning based on peer code review model: Implementation and experience report", *Computers & Education*, vol. 59, pp. 412-422, 2012.
[36] H. Pon-Barry, B. Packard and A. John, "Expanding capacity and promoting inclusion in introductory computer science: a focus on near-peer mentor preparation and code review", *Computer Science Education*, vol. 27, no. 1, pp. 54-77, 2017.
[37] C. Van Bael, "Feature Review Checklist", Retrieved 2018, June 22, from https://www.polteq.com/wp-content/uploads/2016/06/Gherkin-Checklists-1.pdf
[38] G. Khattak and P. Koopman, "Embedded System Code Review Checklist", Retrieved 2018, June 22, from https://users.ece.cmu.edu/~koopman/pubs/code_review_checklist_v1_00.pdf
[39] J. Dalbey, "Code Review Checklist – Java", Retrieved 2018, June 22, from http://users.csc.calpoly.edu/~jdalbey/301/Forms/CodeReviewChecklistJava.doc
[40] M. Ernst, "Code Review Framework", Retrieved 2018, June 22, from https://homes.cs.washington.edu/
[41] C. Marshall and G. Rossman, *Designing Qualitative Research*. SAGE Publications, 2015.
[42] S. Oltmann, "Qualitative Interviews: A Methodological Discussion of the Interviewer and Respondent Contexts", *Forum: Qualitative Social Research*, vol. 17, no. 2, art. 15, 2016.
[43] A. Queirós, D. Faria and F. Almeida, "Strengths and Limitation of Qualitative and Quantitative Research Methods", *European Journal of Education Studies*, vol. 3, no. 9, pp. 369-387, 2017.

Author's Profile

Fernando Almeida is a lecturer at Polythecnic Institute of Gaya and researcher at University of Porto and INESC TEC. He holds a PhD. in Computer Science Engineering and a MSc. in Innovation and Entrepreneurship. His current research areas include software engineering, agile development and innovation policies.

How to cite this paper: Fernando Almeida, " Framework for Software Code Reviews and Inspections in a Classroom Environment", International Journal of Modern Education and Computer Science(IJMECS), Vol.10, No.10, pp. 31-39, 2018.DOI: 10.5815/ijmecs.2018.10.04

