Attaining competences in software quality oriented design based on cyclic learning

Camelia Şerban and Virginia Niculescu and Andreea Vescan
Faculty of Mathematics and Computer Science
Babeş-Bolyai University, Cluj-Napoca,România
{camelia, vniculescu, avescan}@cs.ubbcluj.ro

***Abstract*—This Research to Practice Full Paper delineates the impact of using cyclic learning to obtain competences in software quality oriented design.**

Nowadays, the need for quality in the software systems has become more and more a concern for many researchers and industry practitioners. Developing students' appropriate competencies and skills in writing quality programs must be an important objective of any Software Engineering related course from the Computer Science Curricula. In order to attain this goal, the paper presents a new strategy for reflecting software quality models into Software Engineering related courses based on cyclic learning. The method is based on an educational strategy that integrates the cyclic learning approach and induces to the students the awareness regarding the importance of developing quality software. We focus on a set of software quality characteristics described by the ISO25010 quality model for which we analyze the level of knowledge attained by the students during the entire bachelor cycle of studies. The study is directed by a detailed analysis of three courses: Advanced Programming Methods, Parallel and Distributed Programming, and Software Systems Verification and Validation, which were chosen in order to master the analysis complexity, but at the same time to assure coverage of as many quality attributes as possible.

The investigation includes qualitative and quantitative analysis, directed by the objective of establishing the efficiency and effectiveness of the approach. The results obtained confirm both students' awareness regarding the importance of learning software quality attributes, and the efficiency of using cyclic learning in teaching this subject. We also outline several insights and advantages, and we conclude by showing that the proposed strategy fulfilled the expected objectives.

***Index Terms*—cyclic learning, quality attributes, high performance, testing, undergraduate, IT industry.**

I. Introduction

Programs have continuously increased in size and complexity leading to higher development costs and lower productivity. Software systems have become inflexible (difficulty in adding new functionality), monolythical (provided functionality not based on components), and difficult to maintain (any change comes with an unending chain of adjustments in multiple places). As a consequence to the development above, the need for quality in the software systems has become more and more a concern for many researchers and industry practitioners. Also, developing students' appropriate competences and skills in writing quality programs must be an important objective of any Software Engineering related course from the Computer Science Curriculum.

In order to attain this goal we present a methodology/strategy based on cyclic learning, adopted at our faculty, through which the students are made aware about the need for creating quality software even from the early courses of the curricula. We describe in this paper this methodology/strategy and our studies that reflects its necessity and also the associated results.

For outcomes analysis of the approached strategy, we focused on a set of software quality (SQ) attributes for which we analyse the level of knowledge attained by the students during the entire cycle of studies. The paper takes into account only a subset of the SQ attributes from ISO2010 quality model [21] based on their link to the considered courses objectives.

In order to evaluate the proposed strategy we formulate the following research questions:

RQ1: What is the impact of using cyclic learning to obtain competences in software quality oriented design?

RQ2 What is the impact on students' awareness as regards the teaching of software quality?

The paper is organized as follows: the next section briefly describes the cyclic learning approach, Section III sets the context by outlining the software quality model and the objectives of the analysis, Section IV presents how the teaching of the specific characteristics of software quality are approached in our faculty, and Section V describes the conducted analysis and results. The conclusions are emphasized in Section VI.

II. Cyclic Learning

The teaching approach based on *the learning cycle* uses three distinct phases of instruction [24]: *the exploration* (which provides students with firsthand experiences with science phenomena), *the concept introduction* (which allows students to build understanding of science concepts), and *the concept application* (which requires students to apply their understanding to situations or new problems).

Educational taxonomies are useful tools in developing learning objectives and assessing student attainment. The most widely cited in the literature is the Bloom's taxonomy [8]. Bloom's taxonomy has six categories, where each category builds on the lower ones: Knowledge, Comprehension, Application, Analysis, Synthesis Abilities and skills, and Evaluation.

Bloom's taxonomy has been revised by Anderson et al [25], [5], which changed the nouns listed in the Bloom's model into verbs, reversing the order of the highest two levels. A key difference between the revised taxonomy and the original taxonomy is that the type of knowledge elements are also defined: Factual knowledge, Conceptual knowledge, Procedural knowledge, and Metacognitive knowledge. This provides a matrix into which learning objectives are mapped.

The revised Bloom's classification [8] defines six categories: *Remember*, *Understand*, *Apply*, *Analyze*, *Evaluate*, *Create*.

The implied way of teaching is very similar to the spiral process of teaching which is described in [19] in correlation to Bloom's taxonomy. This taxonomy is considered appropriate for Computer Science teaching.

Simplified variants of Bloom's taxonomy have been considered, either by grouping each two neighbor levels into one [35], or by extracting the most influential three categories [20].

This last variant is also in direct correlation to ACM classification of the level of knowledge [1]:

1) **K** = Know the term (˜ ACM: **Familiarity**);
2) **C** = Comprehend so as to illustrate (˜ ACM: **Usage**);
3) **A** = Apply it in new context (˜ ACM: **Assesment**).

The advantage of using cyclic learning approach lies in the fact that the students return to previously learned concepts with regularity, and each time they have the opportunity to extend and deep in their knowledge related to them. Thus, in order to learn new concepts, programming methodologies, principles, and rules, we resume those already studied and try to understand the need for new ones.

III. Setting the context

The current section outlines the main concepts and elements of our research, illustrating the software quality model and the objectives along with the curriculum descriptions.

A. Software Quality Model

There are many factors that decide the quality of a software system. Several quality models [22], [23], [21] have been developed since the early 1970.

The current paper refers to the ISO25010 Quality Model (ISO25010) [21] which defines eight quality attributes:

- *Functional suitability*: "the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions."
- *Performance efficiency*: "the performance relative to the amount of resources used under stated conditions."
- *Compatibility*: "the degree to which a product, system or component can exchange information with other products, systems or components."
- *Usability*: " the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."
- *Reliability*: "the degree to which a system, product or component performs specified functions under specified conditions for a specified period of time."
- *Security*: "Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization."
- *Maintainability*: "the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements"
- *Portability*: "degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another."

Each SQ attribute, comprised in ISO 25010 quality model [21], includes a set of sub-characteristics that specialize the quality analysis.

B. Objectives and curriculum description

A very important objective in achieving good competences in building high quality software is to analyse how to design the lectures and the learning activities in order to increase the level of students' abilities in writing programs that meet SQ attributes.

The proposed bachelor educational strategy oriented on SQ is characterized by the following:

- Use cyclic learning for introducing SQ standards by spreading the knowledge throughout the entire curriculum.
- Let the students know about SQ standards from early courses in order to follow and reinforce them when they learn about different technologies and methodologies.
- Increase the level of interest of the students in learning software characteristics by inform them that they are part of SQ standards – explicit approach.

The objective of the current study is to analyse this strategy and answer to the research questions specified in Section I in order to evaluate the outcomes.

All the Software Engineering (SE) associated courses that are part of the Computer Science Curriculum from our faculty discuss and focus on SQ characteristics, each course being oriented more on some of the SQ attributes and less on the others. Thus, for example, starting from the first year of study at the courses “Fundamentals of programming” (FP) and then at “Object oriented programming” (OOP) the students learn to develop applications by using incremental development.

In pursuance of evaluating the outcomes of the SQ oriented edu-strategy, we have analysed the courses that are connected to SQ: Fundamental of Programming (FP), Operating Systems (OS), (Data Structures and Algorithms (DSA), Object-Oriented Programming (OOP), Advanced Programming Methods (APM), Software Engineering (SE), Systems for Design and Implementations (SDI), Parallel and Distributed Programming (PDP), Software Systems Verification and Validation (SSVV).

To master the implied analysis complexity, from all these courses we have selected three courses for a detailed analysis: “Advanced Programming Methods” (APM), “Parallel and Distributed Programming” (PDP), and “Software Systems Verification and Validation” (SSVV). Choosing these courses was determined by the degree on which the SQ attributes are emphasized, but also on the ability to analyze a set as large as possible of SQ attributes. Figure 1 presents the entire package of SE courses, emphasising the three selected courses.

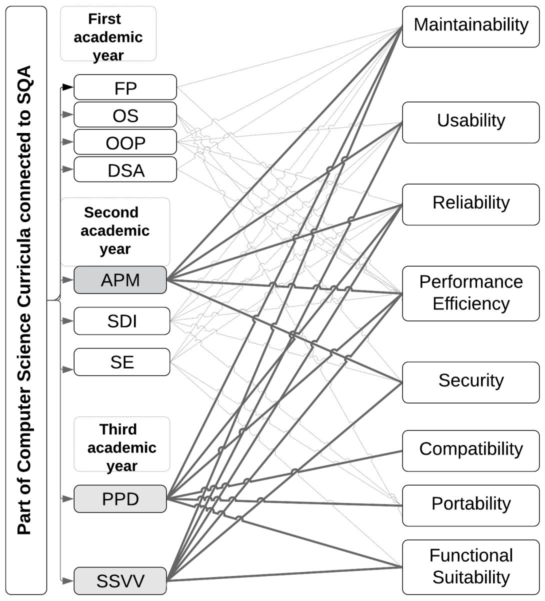

Fig. 1. SQA reflected into Software Engineering related courses

IV. Software Qualities Attributes Reflected into the Bachelor Courses

The current section describes the way of which Software Qualities Attributes (SQA) are reflected in the selected courses AMP, PDP and SSVV.

For each course we aim to emphasize the way in which we address the research questions stated above regarding the quantitative and qualitative analysis.

A. SQA Reflected into APM course

It is well known that a good internal structure (design) of software system has a strong positive impact on its quality attributes [42]. For instance, reliability seems to be related only to software system external behaviour. However, *failures* are usually derived from *faults* or *design flaws* within the system internal structure [43]. In this context, we study the software quality attributes by making reference to the internal structure of the system or, in other words, to the system design and architecture.

Figure 2 expresses the fact that software quality attributes are in fact external characteristics of the system that are are influenced by the system internal structure (design), expressed in terms of design principles, heuristics and rules.

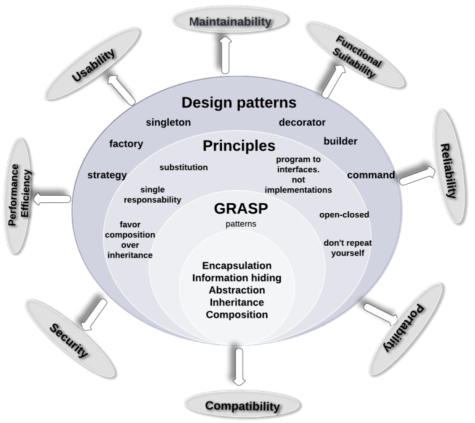

Fig. 2. Good internal structure $\longrightarrow$ Good external quality

In what follows we aim to presents the way in which quality attributes comprised in ISO25010 model are reflected into APM course. This is better emphasized in the lab project developed by students during laboratory classes throughout the entire semester as part of their summative and formative assessment.

The lab project development is split into iterations, each iteration being defined by a set of functional and non-functional requirements. The *Non-functional requirements* are those related to *software system internal quality* and are expressed in terms of design principles, heuristics, rules and architecture. An example of how SQ attributes discussed at APM course are connected to lab project requirements is presented in a graph described in Figure 3. These requirements are detailed in order to offer a scoring scale for lab instructors regarding the considered SQ attributes. In what follows, we present here the non functional requirements for the first iteration of the project.

Iteration 1 - Non-functional requirements related to maintainability

- Design Architecture (3 tier): persistence layer, service layer, user interface (ui) layer.

- Data persistence: in memory and in file. You will have to use the given Repository interface (no changes allowed there); avoid code duplication by using custom generic classes and methods.
- Service: one service for each entity.
- User interface design: console based.
- Design pattern: command, decorator, factory method, singleton
- Testing design: Unit tests: all methods will be documented and tested.
- Data validation: strategy design pattern.
- Proper exception handling: exceptions should be wrapped in custom exception classes, thrown further, caught in the ui.
- Refactoring: refactor code whenever duplication appears.

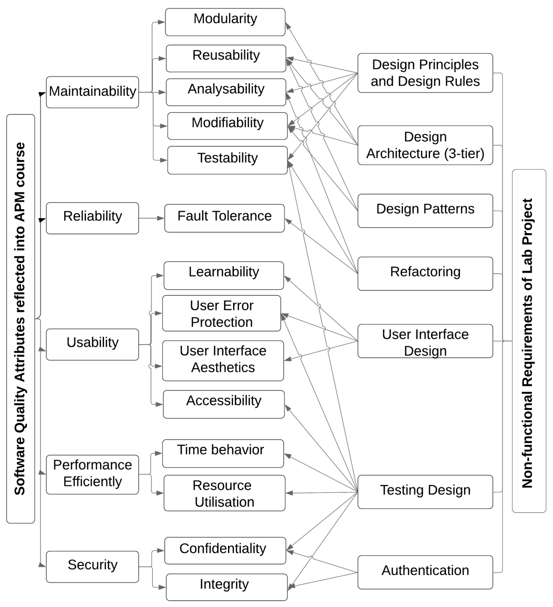

Fig. 3. SQA reflected into APM course

The grades obtained by students for each quality attribute during the semester are both part of students' summative assessment and they are also used to validate our proposed approach, more precisely to address the RQ1 by means of an quantitative analysis.

B. SQA Reflected into PDP and associated courses

Even from the first courses that discuss fundamentals of the algorithms and data structures, the students are learned to think about time-complexity and space-complexity of their algorithms, and the importance of these for obtaining fast and efficient computations. So, the *Performance and Efficiency* SQ attributes are first introduced at these beginning courses.

The main reason of using parallelisation is the performance, and it is applied on a large scale in this multi- and many-cores era. In our curriculum, threads and multithreading computing are introduced very soon, starting with OS course and continuing with OOP and APM courses.

Distributing computing is also related to achieving performance since through this is not only possible to develop client-server or peer-to-peer applications, but also it allows splitting the computation in components that could be computed remotely as in the case of grid-computing or cloud-computing. Distributed computing is introduced starting with the *Network Systems* course, and continues in SDI and PDP courses.

PDP course aims to substantiate the knowledge regarding parallel and distributed computing on shared and distributed memory platforms. Besides the multi-threading reinforcement, the students are introduced to new paradigms and technologies as - MPI (Message-Passing-Interface) OpenMP, or CUDA.

The course also presents the specific design patterns of parallel and distributed programming. This facilitates developing the ability of creating well-structured applications that offer besides good performance also good maintainability.

The performance is analysed both through theoretical and empirical evaluation. Using theoretical metrics similar to those discussed for the sequential case (as time-complexity) allows students to understand that performance oriented analysis could be done even from the design phase of the parallel programs development. In this context theoretical metrics as: speed-up, efficiency, cost, scalability and granularity are discussed. Reliable measuring of the execution time of parallel programs, implies repeated executions for different settings regarding the problem size of the used resources.

Reliability is also very important for parallel and distributed applications. It is well known that non-determinancy is present in these kinds of applications due to the lack of the control on the order in which operations are done. Assuring correctness is many times a difficult task, and simple testing is for sure not enough. Problems such as deadlock, livelock, or starvation are carefully analysed.

The Figure 4 summarizes how the quality attributes and their characteristics are reflected into PDP course.

C. SQA Reflected into SSVV course

The main topics of the SSVV lectures are: inspection/review [27], testing with test design strategies using black-box testing and white box testing technique [28], integration and system testing [29], symbolic execution, model checking [30], and correctness [31], [32], [33].

All quality attributes checked in Table I are discussed during lecture hours. Most of them are discussed during seminars and laboratories with concrete examples, and practiced during laboratories. Several of the quality attributes are only presented, discussed and exemplified during lecture hours.

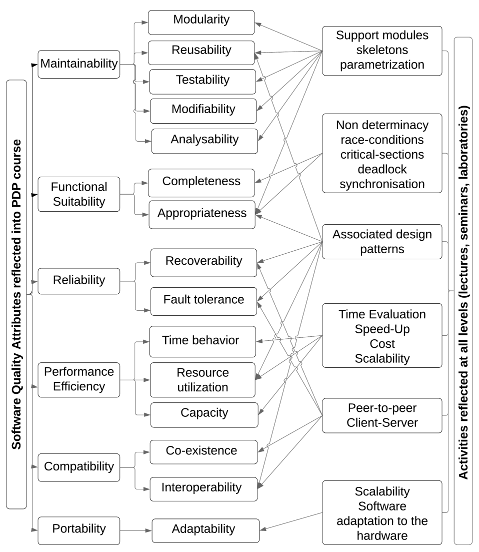
Software Quality Attributes reflected into PDP course
Maintainability
Functional Suitability
Reliability
Performance Efficiency
Compatibility
Portability
Modularity
Reusability
Testability
Modifiability
Analysability
Completeness
Appropriateness
Recoverability
Fault tolerance
Time behavior
Resource utilization
Capacity
Co-existence
Interoperability
Adaptability
Support modules skeletons parametrization
Non determinacy race-conditions critical-sections deadlock synchronisation
Associated design patterns
Time Evaluation Speed-Up Cost Scalability
Peer-to-peer Client-Server
Scalability Software adaptation to the hardware
Activities reflected at all levels (lectures, seminars, laboratories)

Fig. 4. SQA reflected into PDP course

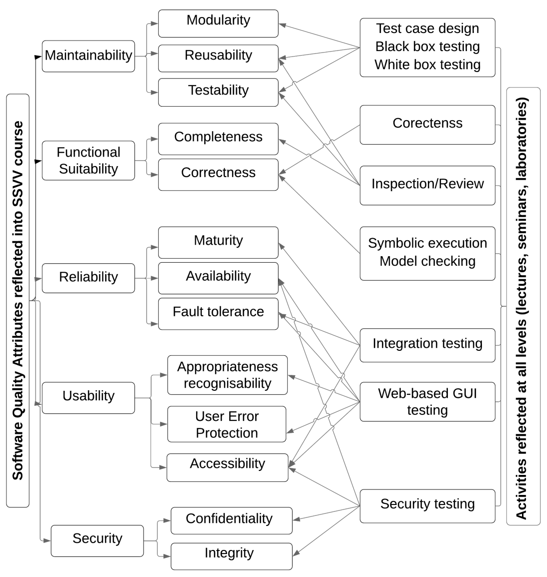

Fig. 5. SQA reflected into SSVV course.

It is worth mentioning that the seminars and laboratories are performed in tandem, i.e. the same topic being addressed in the same weeks period for both seminar and laboratory, as further exemplified:

- **Inspection (Sem. 1, Lab. 1)** - to learn information about the Software Under Test (SUT) and decide the difficulty level for carrying out testing activities on that artefact (specification document, design document, source code).
- **Black-box testing** - to design test cases based on the specification using Equivalence classes and Boundary value analysis.
- **White-box testing** - to design test cases based on the source code using Control flow Graph and various coverage criteria (statement, decision, condition-decision, multiple condition-decision, independent path, loop).
- **Integration testing** - to experience various integration strategies: big-bang, incremental integration (top down and bottom up) and sandwich integration.
- **GUI testing** - to test web-based application, i.e. using Selenium Web Driver to test the GUI.
- **Correctness** - to exercise proving correctness of algorithms and methods using ESCJava2.

Several of the performed activities during SSVV course are directly and indirectly addressing various quality attributes as stated in Table I. Figure 5 emphasize the connections between the quality models and the activities performed in SSVV course.

V. Analysis

The investigation addresses the research questions formulated in Section I and it was directed by quantitative and qualitative analyses.

First, we analyse the impact of spreading SQA through the courses, based on an analysis done by the professors that adapt their courses; this is a qualitative analysis but it is oriented on answering RQ1. Then, we present a qualitative analysis (oriented especially on answering RQ2), and a quantitative analysis (oriented especially on answering RQ1), which are both based on the students' input.

A. *The impact of spreading SQA through the courses*

The software quality attributes are spread throughout the entire CS curricula from our faculty, but there are some courses that emphasize to a better extent the importance for a software system to have a good quality design. These courses are mainly software engineering related ones, but there other courses that also discuss these SQ related aspects, too.
The novelty of the analysed approach of introducing SQ in CS curricula stands in emphasizing explicitly the SQ attributes even from the first courses in order to make the students understanding their importance very soon and to achieve competences in building high quality software.

For example, when the students learn about the time-complexity of the algorithms they understand in a certain measure that this is related to the performance of their implementation. As well, space-complexity analysis gives them a first understanding about efficiency. But if the concept of *Performance and Efficiency* is introduced as a SQ attribute, as an essential feature in developing efficient applications, the students could make a better connection to its practical importance and could understand the relation with the other performance and efficiency related mechanisms and principles that appear in the following courses.

Similarly, when code reusing is discussed in the object-oriented context, introducing *maintainability* as a SQ standard, facilitates a better understanding of the need for modularity and testability. Also, code reusing is discussed early in the context of design pattern (a solution for a recurrent problem that appears in different contexts), giving a name and a solution skeleton for that problem proved to be essential in creating good abilities in software development. These SQ attributes are first introduced at FP course from the first year of study and leaded to a more advanced level to the others courses from the chain of Software Engineering related ones.

SQA	Subcharacteristics	**APM**	**PDP**	**SSVV**
Maintainability	Modularity	A	A	A
	Reusability	A	A	A
	Analysability	K	C	
	Modifiability	K	C	
	Testability	K	C	C
Functional suitability	Completeness	K	K	A
	Correctness	C	C	C
	Appropriateness	K	C	C
Reliability	Maturity	K	K	C
	Availability	K	K	C
	Fault tolerance	C	C	A
Usability	Appropriateness recognisability		K	
	Learnability	K		
	User Error protection	K		C
	UI aesthetics	C		
	Accessibility	K		K
Performance Efficiency	Time behavior	K	A	
	Resource utilisation	K	C	
	Capacity		K	
Security	Confidentiality	K		K
	Integrity	K	K	
	Non-repudiation			
	Authenticity	K		
Compatibility	Co-existence		C	
	Interoperability	K	C	
Portability	Adaptability		K	

TABLE I
SQ attributes reflected into analysed courses

Corroborating all these analyses to the involved professors' evaluations, we obtained a global overview of the level of the competences for the considered SQ attributes. This is provided in Table I, where for each attained subcharacteristic and for each course the K, C, A levels, as discussed in Section III, are provided. The effectiveness of cyclic learning approach is emphasized (RQ1), and the fact that, at the end of the curriculum cycle, for many SQ attributes levels A and C are obtained, signifies that we are close to the fulfilment of the objectives.

The benefit has been emphasised in the medium or complex projects that the students developed for *Collective Project* discipline and for the graduation diploma project. The new approach leaded to an increase of the requirements related to software quality of the students' applications, but they were fulfilled without difficulty. This has been emphasized by the industry feedback that was received through the firms that are interested in evaluating the student graduation applications, too.

B. *Qualitative Analysis*

1) APM: At the APM course the students were asked to complete a questionnaire related to 4 of the 5 SQ attributes that were linked to APM course. The number of students that completed the questionnaire was 102. For each SQ attribute the students have to estimate their knowledge level from Bloom's taxonomy regarding that attribute, before they followed the APM course and after they finish this course, at the end of the semester. The questions addressed to students were: *What was/is in your perception, the level of knowledge (K, C, A) you acquired regarding attributeX, before and after APM course?*, where *attributeX* $\in$ {*Maintainability, Reliability, Usability and Performance Efficiency*}. The results showed in Figure 6 reveals that there is a significant difference between the students' learning level at the beginning of the semester and the corresponding level of learning at the end of the semester, for all the considered SQ attributes.

Maintainability is the SQ attribute that has the highest increasing in students' learning level for A level (57 students) and few students are at K level after completing the course (5 students). This is justified by the fact that even from the first year of study, students are asked to design programs that are reusable, modular, easy to modify and test. After the maintainability attribute, reliability and usability attained also a high level of students' learning; more accent was put on testing and usability starting from APM course. There is an extended lab project where the usability quality attribute has the biggest weight.

Regarding performance and efficiency, there is also an increasing in the number of students that evolve from K to C and from C to A levels. This quality attribute is discussed more at the PDP course where students reach the highest level of knowledge.

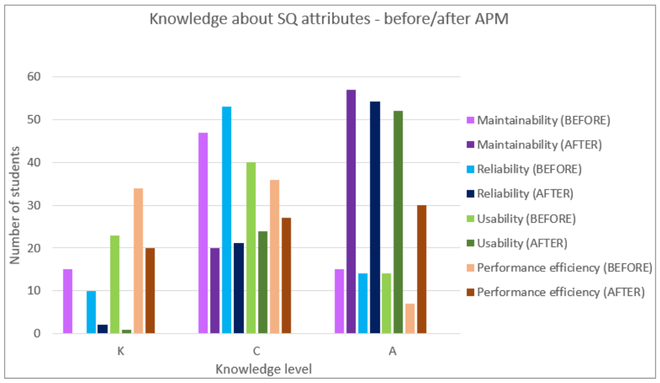

Fig. 6. APM Qualitative Analysis

Beside the questions connected to students' knowledge level regarding SQ attributes, there was an open question: "What is in your opinion the effectiveness of the proposed cyclic learning approach in teaching software quality?". One response is worth to be mentioning here: "I felt myself saved when I have seen that the APM course revise some important OOP concepts and mechanisms, together with principles and rules for good object-oriented design, and it reinforced the aspects related to software quality."

2) PDP: At the PDP course the main SQ attributes that are discussed and enforced are *Performance and Efficiency*. This is not a completely new concept for the students, since also for this, cyclic learning approach was used. During PDP course, the students are requested to evaluate the performance by measuring the execution time, to investigate different approaches for optimisation, but also to associate different design approaches to performance. Achieving performance using well-defined or predefined skeletons, or by applying parallel design patterns are investigated, while the advantages brought by these in assuring *Correctness* and productivity are analysed, as well. So, connections between *Performance* and other SQ attributes are emphasised, too. These are theoretically discussed, but also applied in the context of building the laboratory applications, as well as in creating a framework for their testing.

The practical works are done on different hardware systems: starting from personal laptops to high performance cluster infrastructures. This facilitates the students' understanding of the need for *scalability*, but also the constraints given by the available resources.

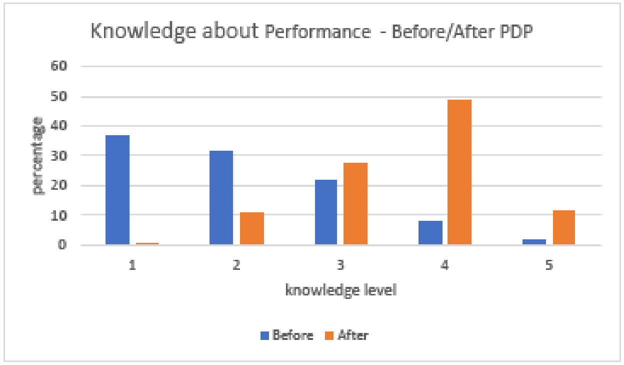

Fig. 7. Evaluation of the knowledge related to the performance evaluation and importance, before and after the PDP course (student auto-evaluation).

In order to obtain information based on which we can provide answers to the research questions RQ1 and RQ2, a questionnaire was given to students (165 responses were received); this questionnaire evaluates their self assessment regarding the application of cyclic-learning approach, and also about how important is the performance in the application development. Many interrogations about their auto-evaluation of the knowledge related to parallel and distributed programming were included, but we emphasize here only the answers related to *Performance* SQ attribute: *"In your perception what was/is your level of knowledge regarding the performance driven development, before and after PDP course"* (Figure 7). In addition, associated comments were registered, and these emphasize a high level of awareness relative to the importance of performance as a SQ attribute. Such examples are: "The required improvements of the client-server application led us understanding the importance of the performance oriented design"; or "The iterations through which the performance of our program has to be improved makes us more carefully to this aspects of the application development", or "We intend to apply performance orientation design in the developing of the graduation project."

3) SSVV: The students answered to a questionnaire at the end of the semester with similar quality attributes related questions as in the APM course, and the result are as expected, with an increasing rate related to Bloom's taxonomy. Thus, we decided not to display the graphic. The number of students that completed this questionnaire was 99. Besides the main questions, another interesting question was *"What topics did you liked more?"*. By mapping the topics and activities with the quality attributes we may consider the liked topics as attained by the students since they perceive it as being understood. The students liked the topics related to testing with various flavours (black-box testing, white-box testing, integration testing, GUI testing), thus the usability and maintainability quality attributes were most appealing. Few students liked correctness and symbolic execution topics, thus functional suitability was the least liked attribute.

C. Quantitative Analysis

1) APM: The results obtained by the students regarding the lab project non-functional requirements connected to SQ attributes are also used to validate our proposed approach. In Section IV-A we have specified how we linked lab requirements with SQ attributes. The considered quality attributes for this analysis are the same as those referred for the qualitative analysis regarding APM course: maintainability, reliability, reusability and performance efficiency. For each of these SQ attributes the students received a grade during the semester, as part of their summative assessment. Also, an additional discussion take place with the students and lab instructor, during the laboratories classes regarding the way in which these requirements are fulfilled. These discussions help course coordinator to improve the learning methods at this course. At the same time, the discussions offer valuable feedback for students, comparing various design/implementation solutions from the perspectives of such quality characteristics.

The results obtained by students regarding the assessment for the considered SQ attributes are presented in Figure 8. The number of students that were assessed is 179. Analyzing further these results we can conclude that there is a significant percentage of students, 90%, having the grade greater then 7, 50% of the students obtained 10 grade for Maintainability quality attribute, and 45% of the students obtained 10 grade for reusability.

2) PDP: At the PDP course the students' evaluation is not explicitly driven by the software quality attributes,

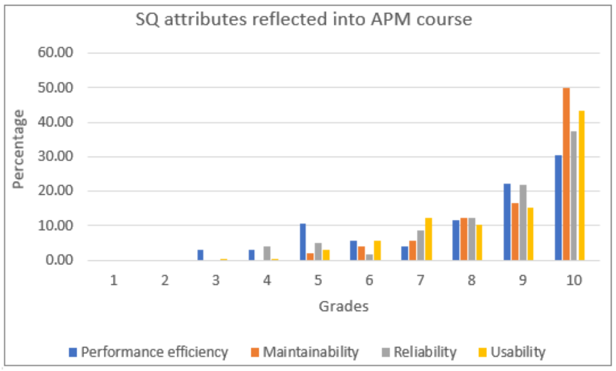

Fig. 8. APM Quantitative Analysis

but they are explicitly enforced to follow them in the software analysis. The obtained grades for the laboratory work reflect especially the performance attribute, but also reliability, maintainability, portability, and the other SQ discussed attributes, since they were based not only on small applications, but also on medium to complex ones. Figure 9 presents the graphic of the results of 196 students for: a client-server application development that reflects in great measure maintainability, a practical test oriented on achieving performance through multithreading, and a written exam that evaluates aspects related to the performance theoretical evaluation.

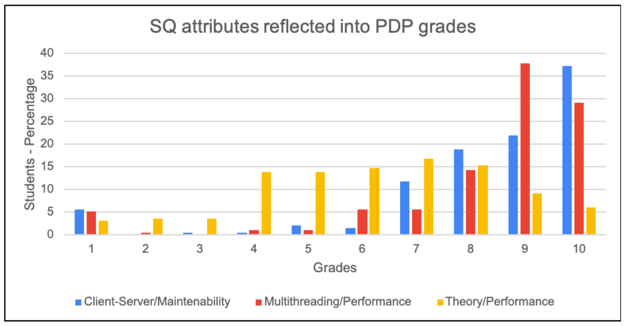

Fig. 9. PDP Quantitative Analysis

3) SSVV: We outline here the results obtained by investigating the obtained grades at different levels, laboratories, seminars and activities during lectures hours. We computed the percentage for each possible grade and for each investigated quality attribute. For *Usability* and *Reliability* students obtained the highest grades. For *Maintainability* the students obtained various grades level, from the lowest to the highest grade, majority of them obtaining from 4 to 6. Details of the obtained results are provided in Figure 10.

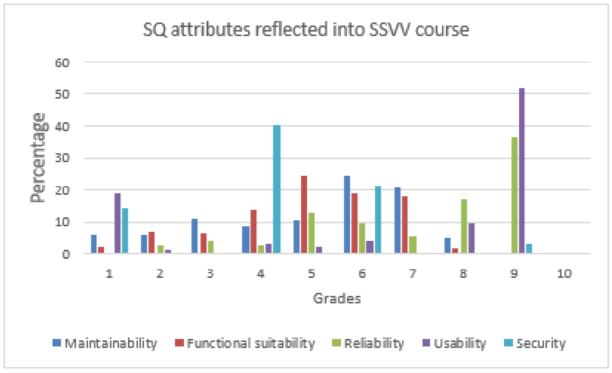

Fig. 10. SSVV Quantitative Analysis

D. Responses for the Research Questions

Considering the above observations and argumentation in the previous section, examining the spread of SQA through the investigated courses and presenting the quantitative and qualitative results for each course, we aggregate the results, outlining the responses for the research questions from Section IV.

Response for RQ1. Cyclic learning approach is effective and efficient in teaching software quality-based design. The results obtained by our analysis acknowledge the effectiveness and efficiency of cycling learning approach, the concepts being first taught using simple terms, then at the later courses being used, and finally being applied in new contexts.

Response for RQ2. The analysed edu-strategy has a positive impact in obtaining competences in software quality oriented design, the students becoming more and more aware of the importance of applying a good design, and of the necessity to obtain these competences as future software engineers. All these were reflected into the improved approaches that they follow in the development of their applications.

VI. Conclusions

The aim of this paper was to investigate the use of cyclic learning methodology in teaching software quality oriented design. The cyclic learning is briefly presented, followed by the used context, i.e. teaching and attaining competences in designing high quality software.

The paper describes the way software quality attributes are introduced in students curriculum during their entire academic program. It also lays out the students awareness regarding the importance of studying how to write quality software. The students will thus be more cognizant and informed about the benefits of applying best practices to obtain a high qualitative system.

The outcome of teaching and learning about quality attributes and designing best qualitative applications have implications to both students and industry: on one side, the students are more aware of the importance of such attributes by applying various design principles and comparing various design/implementation solutions from the perspectives of such quality characteristics, and on the other side, the industry will get "better" work force, specialists in computer science as design architects.

References

[1] *** *ACM Curricula 2013 Report.* https://www.acm.org/education/CS2013-final-report.pdf. (2013), pp 144- 154.

[2] Agile Alliance. *Manifesto for Agile Software Development.* http://agilemanifesto.org/, (2001).

[3] American Psychological Association. *Publication manual of the American Psychological Association (6th ed.)*. American Psychological Association, Washington, DC, US, 6, (2010).

[4] Astels, D. R. *Test-Driven Development: A Practical Guide.* Prentice Hall, 2003.

[5] Anderson, L.W. and Krathwohl, D.R. & all . *A taxonomy for learning, teaching, and assessing: A revision of Bloom's Taxonomy of Educational Objectives.* (Complete edition). New York: Longman.(2001).

[6] Ballone Lena and Duran E.. *The 5E Instructional Model: A Learning Cycle Approach* The Science Education Review, 3(2), (2004).

[7] Beck, K. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, (1999).

[8] Bloom, B. S., Engelhart, M. D.; Furst, E. J., Hill, W. H.; Krathwohl, D. R. *Taxonomy of educational objectives: The classification of educational goals.* Handbook I: Cognitive domain. New York: David McKay Company.(1956).

[9] Booch, G., *Object-Oriented Analysis and Design with Applications.* Benjamin Cummings, Redwood City, 2 edition, 1994.

[10] Cohen, J., *Things I have learned (so far)*, American Psychologist, 45(12), 1304-1312 (1990).

[11] Dyer, C., *Beginning Research in Psychology: A Practical Guide to Research Methods and Statistics*, ISBN: 9780631189299, Publisher Wiley pp. 482 (1995)

[12] Eric E., *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison Wesley, (2003).

[13] Beck, K, *Test Driven Development: By Example. Addison-Wesley Longman, 2002.*

[14] Fowler, M. *Refactoring. Improving the Design of Existing Code. Addison-Wesley, 1999.*

[15] Fowler, M. et.all. *Patterns of Enterprise Application Architecture, Addison-Wesley Professional; 1 edition (November 15, 2002).*

[16] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing, 1995.*

[17] Kleinberg and Tardos *Algorithm Design. Pearson Educational, 2014.*

[18] Fichman, R.G. and Moses, S.A. *An Incremental Process for Software Implementation*, Sloan Management Review, No. Winter, (1999), pp. 39-52.

[19] Fuller, Ursula and all. *Developing a computer science-specific learning taxonomy.* ACM SIGCSE Bulletin 39(4):152-170. (2007).

[20] Huitt, W. *Bloom et al.'s taxonomy of the cognitive domain.* Educational Psychology. (2011).

[21] ISO25010, *ISO25010 description information.* https://iso25000.com/index.php/en/iso-25000-standards/iso-25010, https://www.iso.org/standard/35733.html, (2019), [Online; accessed 30-May-2019]

[22] J.A. McCall and P.K. Richards and G.F. Walters. *Factors in Software Quality*, Griffiths Air Force Base, N.Y. Rome Air Development Center Air Force Systems Command, 1977. @

[23] ISO9126, *ISO9126 description information,ISO 9126-1:2001-Software engineering - Product quality,*

[24] Karplus, R., and H.D. Thier. 1967. A new look at elementary school science. *Chicago, Rand McNally and Company, 1967. pp. 204.*

[25] Anderson, L. W., Krathwohl, D. R. A taxonomy for learning, teaching, and assessing*, Abridged Edition. Boston, MA: Allyn and Bacon, 2001.*

[26] Kramer, J. Is abstraction the key to computing? *Communications of the ACM 50, 37-42, (2007).*

[27] Pezze, M., Young, Y., Software Testing and Analysis: Process, Principles and Techniques *, Wiley, 2007*

[28] Myers, G., The Art of Software Testing*, John Wiley and Sons, Inc., 2004*

[29] Collard, J.-F., Burnstein, I., Practical Software Testing*, Springer-Verlag New York, Inc., 2003*

[30] Baier, C., Katoen, J.-P., Principles of Model Checking*, The MIT Press, 2008*

[31] Floyd, R., Assigning meanings to programs*, In Proceedings of Symposia in Applied Mathematics, pages 19–32, 1967.*

[32] Hoare, C. A. R., An axiomatic basis for computer programming*, Commun. ACM, 12(10):576–580, 1969.*

[33] Dijkstra, E. Guarded commands, nondeterminacy and formal derivation of programs*, CACM, 8(18):453–457, 1975.*

[34] Palmer, S. R., Felsing, J. M., Practical Guide to Feature-Driven Development*, Prentice Hall, 2002.*

[35] Schaffer, H. E., Young,K. R., Ligon, E. W., Chapman, D. D. Automating Individualized Formative Feedback in Large Classes Based on a Directed Concept Graph. *Frontiers in Psychology, 8, (2017) pp. 1-11.*

[36] Shalloway, Alan; Trott, Jim Design Patterns Explained. *ISBN 978-0321247148., pp. 133. (2004)*

[37] Vahid Garousi and Michael Felderer and Feyza Nur Kilicaslan, What we know about software testability: a survey.*ArXiv (2018)*

[38] A. Gonzalez-Sanchez, E. Piel, H. G. Gross and A. J. C. v. Gemund, Minimising the Preparation Cost of Runtime Testing Based on Testability Metrics*, IEEE Annual Computer Software and Applications Conference, Seoul, 2010, pp. 419-424*

[39] K. Saleh, Testability-Directed Service Definitions and Their Synthesis*, International Phoenix Conference on Computers and Communication, Scottsdale, AZ, USA, 1992, pp. 674-678.*

[40] H.-G. Gross, Measuring Evolutionary Testability of Real-Time Software*, PhD thesis, University of Glamorgan, 2000.*

[41] Wagner, S. (2013). Software Product Quality Control*. Berlin: Springer.*

[42] Fenton, N, Software measurement: a necessary scientific basis*, IEEE Transactions on Software Engineering, Vol 20 (3), pp.199-206, 1994.*

[43] ,Sommerville, I., Software Engineering*, Pearson India; 10th edition, 2018.*