Construction of a syllabus adhering to the teaching of software testing using agile practices

Isaac Souza Elgrably
Graduate Program in Computer Scienсe (PPGCC), Institute of Exact and Natural Sciences (ICEN)
Federal University of Pará (UFPA)
Belém – PA, Brazil
isaacelgrably@gmail.com

Sandro Ronaldo Bezerra Oliveira
Graduate Program in Computer Scienсe (PPGCC), Institute of Exact and Natural Sciences (ICEN)
Federal University of Pará (UFPA)
Belém – PA, Brazil
srbo@ufpa.br

***Abstract*— This Research to Practice full paper presents an investigation into the construction of a software testing syllabus, using agile practices as support to make it adherent to teaching by academic skills. In most Brazilian universities, software testing is approached mainly as subtopics of the subjects of Software Engineering or Software Quality; this little contact with the topic can become a discouraging problem for higher education students seeking knowledge about this important area. For the development of the syllabus, we had as reference the competence and syllabus guidelines of the Ministry of Education of Brazil, which is the regulator of the syllabus guidelines for undergraduate and graduate courses in computing in Brazil. In conjunction with these guidelines, other assets in the area were aligned, such as the academic syllabus of the Brazilian Computer Society and the ACM / IEEE, in conjunction with the SWEBOK test-oriented knowledge areas, the Test Maturity Model Integration process areas and guided by agile practices, principles and techniques found in the literature. After this stage, the construction of teaching units for teaching of software testing was carried out. These teaching units were evaluated as part of the software testing teaching model by means a focus group, composed of three doctors who are also specialists in the teaching of software engineering, with experience in teaching and research. Thus, this work also provides a set of skills that we consider favorable to the teaching of software testing, considering several academic syllabuses analyzed. Academic program managers and teachers can use the knowledge generated in this article to facilitate the construction of syllabuses or subjects related to tests, learning objects and academic syllabus. Our findings can also be learning facilitators for competency-based teaching and learning experiences for testing professionals.**

Keywords—*syllabus, testing teaching, software testing, agile methods*

I. Introduction

For a long time, the software testing task was considered a punishment for programmers. Testing was considered an ungrateful task, because developers were just expected to build good quality software [1]. Currently, agile testing is a testing practice that follows agile principles, characterizing testing from the user's perspective, focusing on the testing capabilities in iterative cycles, rather than emphasizing the rigorous testing phase of traditional testing [2].

Software testing is contained in the Software Engineering (SE) area. According to the syllabus [3][4], we can see it as one of the topics in the Software Engineering, from the SWEBOK knowledge guide [5]. Pressman [6] states that testing often requires more design work than any other action in Software Engineering. If done casually, unnecessary time and effort is wasted. With the rising need for software with higher quality and a lower margin of error, there has been a marked growth of researchers and developers in software testing area [7]. Agile methods consider software testing to be a critical activity, indicating that test cases should be written before the software units they intend to test [1].

Regarding the syllabus content and subjects that served as a reference for the construction of this syllabus, we have the National Curriculum Guidelines approved by resolution No. 05 of 2016/16/11 [8]. For the composition of academic syllabus, the training reference guide for undergraduate courses in computing [9] and the ACM / IEEE guide [10] were used as a basis. In mid-2014 ACM / IEEE created SWEBOK, which aims to help organizations to have a consistent view of Software Engineering, helping to characterize its content [5]. In the context of software quality control, Test Maturity Model Integration (TMMI) was used [11]. As a last support, a set of agile practices, principles and techniques was also used, such as, the Agile Test Manifesto [12].

This syllabus construction will also present some results of the Teaching Mapping or application of tests with a focus on agile methods, which was made from an asset mapping [13], which analyzed several components of software tests that serve as frameworks for application in industry or teaching a particular area.

Even nowadays there is a considerable deficiency in the teaching of software testing, there are still complaints related to the teaching of software testing in education, because its focus is on theory and lack of practice to show students how the concepts should be applied [14]. Depending on how the syllabus is designed, students graduate without any real testing practice [14].

Thus, the following Research Question (RQ): *How can we develop a syllabus that adopts teaching guidelines for software testing in the Brazilian context that develops knowledge and skills for students that may be relevant in the software industry?* To answer this question, a syllabus focused on teaching of software testing was built, based on results obtained from a specialized review of the literature and a mapping of test assets [13], and subsequently evaluated by an expert panel. The result is software testing teaching syllabus that led to the construction of a set of teaching units supported by agile practices. Instructors and professors who develop courses and subjects with topics in the software testing can use these results.

So, the goal of this syllabus is to try to consolidate a form of syllabus construction that focuses on student teaching and learning, that aims to consolidate a way of applying approaches closer to the challenges that students may encounter in the labor market, and assist in the teaching-learning process of building of subjects or teaching of units related to the software testing. Another goal is to make the syllabus adaptable so that it is not only specifically defined for use in an undergraduate course in Computer Science and in a Software Engineering subject, but also for different undergraduate courses in the computing and even serves as training for software development teams who would like to acquire software testing skills and abilities.

In addition to this introductory section, this paper is structured as follows: in Section II the theoretical basis used for building the syllabus is provided, in Section III the research methodology is described including how the research was organized and what results were sought, Section IV presents how the mapping that gave rise to the assets used for the construction of the syllabus was designed, Section V presents the evaluation of the syllabus that took place in the form of an experts panel, finally, we conclude by presenting the contributions of this syllabus, in addition to presenting its limitations and possible threats to validity and highlighting our plans for future studies in Section VI.

II. Background

This section presents the basis used to provide a theoretical basis for the concepts used in this work.

A. *Teaching of Software Testing*

Despite its relevance, studies mention that graduates of Computing courses graduate with limited knowledge about software testing [15]. In the investigation by Sclaton [14], regarding the knowledge gaps of the experts in software testing, the results indicated a deficiency for all test topics in practical activities.

According to Valle [16], this may reflect the reference syllabus, since universities elaborate the curricular structures of their courses taking into account the reference syllabus proposed by ACM / IEEE and by the Brazilian Computer Society (SBC), and these guidelines recommend that content be addressed as content units in software engineering subjects. What often ends up being insufficient for teaching, mainly in the practical part of teaching of software testing.

The ACM / IEEE guide indicates that, as information technology is a practice-oriented subject, it is important that many professors have practical experience in the main information technologies and also adds that for many subjects the practical part of applying information technology is essential for career success [10]. Thus, for the creation of this syllabus, these problems found in the literature were taken into account.

B. *Syllabus Guidelines of Ministry of Education in Brazil (MEC)*

In Brazil, undergraduate courses need to be conducted through the National Syllabus Guidelines, which have the mission of guiding and offering solid basic training, preparing future graduates to face the challenges that exist in society, in the labor market and to provide conditions for professional and academic practice. The process of regularization of the syllabus guidelines for the bachelor and degree courses in Computing was approved in 2016, via opinion, by the Ministry of Education in Brazil [8].

The Syllabus Guidelines served as a foundation for the realization of courses in the Information Technology, for the creation of their pedagogical projects in addition to the clear design of the course, with its peculiarities, its curricular matrix and its operationalization [8]. In this work, the MEC syllabus guidelines are the main competencie references for creating the syllabus that is being presented.

C. *Brazilian Computer Society*

The Brazilian Computer Society (SBC) is a non-profit society that brings together students, professor, researchers and professionals in the computing and information technology in Brazil and has as some goals: to improve access to information through information technology, to promote digital inclusion, to encourage research and teaching in computing in Brazil, and to contribute to the training of computer professionals with social responsibility.

SBC members are responsible for discussing how undergraduate courses should be conducted in the last decades, either by setting up committees for the elaboration of Reference Syllabus or by discussing the ways of evaluating these courses by MEC [4].

For this work, the Training Reference Guide was used for guides to Undergraduate Courses in Computing [3][4][9][10]. Reference guides are not syllabus, but reference material for those who are preparing their syllabus. This material must be worked on in conjunction with national guidelines.

D. *ACM / IEEE Guideline*

The ACM / IEEE reference syllabus defines a framework of topics and units of knowledge in order to specify teaching and vocational training guidelines for the study areas of computing. It serves as a teaching base for computer topics for several universities, thus it is recognized and adopted internationally.

The guideline [3] aims to collaborate with realistic and adoptable recommendations that provide guidance and flexibility, allowing for innovative syllabus designs and following recent developments in the field. The guidelines aim to provide clear and implementable objectives, in addition to providing the flexibility that programs need to respond to a rapidly changing field such as information technology.

For this research, this guideline was selected for its high level of detail and its syllabus guidelines were aligned with those of MEC and SBC.

E. *SWEBOK*

IEEE, seeking to accompany the emergence of several study areas and sub-areas in computing, developed the SWEBOK, which is a guide for the body of knowledge in Software Engineering, in order to promote the professionalization of Software Engineering.

According to SWEBOK [5], its five main goals are: (i) promoting a consistent view of software engineering worldwide, (ii) specifying the scope and clarify the location of software engineering in relation to other subjects, (iii) characterizing the contents of the software engineering subject, (iv) providing topical access to a set of software engineering knowledge, and (v) providing a basis for the development of a syllabus, for professional certification and licensing of materials related to software engineering.

The Guide covers the necessary Software Engineering knowledge, but not enough for the Software Engineer, it does not focus on specific subjects such as which programming languages to choose, but on the essential knowledge of the knowledge areas.

For this research, this guide was selected for providing knowledge base regarding the creation of syllabus and being a guide aimed at a wide target audience: public and private organizations, Software Engineers, Authorities responsible for developing public policies, educators and students.

F. TMMI

TMMi (Test Maturity Model integration) aims to support organizations in evaluating and improving the Test Process, addressing important issues for test managers, test engineers and software quality professionals, and applied broadly, with the objective to cover all software product quality and related activities. TMMi has 5 levels of maturity, separated by process areas, these have goals that can be specific or generic. These goals are composed of several practices that are also specific or generic and each of them contains a set of sub-practices, which are responsible for guiding their interpretation and evaluation.

The model was designed to improve processes, being partitioned into five levels at which an organization must prepare to evolve the testing process. The levels lead from an ad-hoc and unmanaged test process to a managed, well-defined, metrical and optimized process [11].

For this research, it was selected because it covers all activities related to the quality of a software product independently of the test that is performed.

G. Agile Methods

Agile methods always strive to try to respond to changes that may come to occur in requirements, integrating processes, under development and testing as simplified processes that focus on accepting changes and adapting [17].

Some characteristics of tests using agile methods are that they are almost always performed by all team members, must occur at all stages of the software process development lifecycle, prioritize whenever possible the use of automated mechanisms and occur frequently in continuous cycles, so that the product or software evolves.

For this research, the use of agile testing practices was selected as they aim to assist development teams in delivering high quality products. However, still maintaining the rigor, discipline and use of refined techniques in carrying out the tests, as there is the challenge of the software, in addition to being functional, being delivered with the shortest possible terms and costs.

III. Research Methodology

To achieve the goals of this research, an analysis was initially carried out in the National Syllabus Guidelines for Undergraduate Courses in Computing, approved in 2016 [8] to identify the important topics and skills for Software Testing. Then, a mapping between several assets was carried out to establish a knowledge base and be used as a reference for the development of this Syllabus proposal.

The other assets are the syllabus [3][4][10] of undergraduate courses in Information Technology and their practices aimed at Software Testing. Subsequently, was analyzed the set of SWEBOK practices, which is an important knowledge guide on the knowledge areas of Software Engineering, and the processes areas indicated by TMMi and finally, the values present in the Agile Test Manifesto were included [12].

Based on the indications found in the mentioned assets, the creation of a syllabus proposal for the teaching of Software Testing was inspired by agile values presented in this paper. The evaluation of the importance and content of the proposal was made through a experts panel composed of three doctors, who work as professors and researchers in the teaching of Software Engineering in different Brazilian educational institutions.

Initially, there was an individual analysis and evaluation of the syllabus proposal, where each expert answered a questionnaire about the content of the proposal. Then, there was a consensus meeting with everyone in order to solve disagreements and capture critical problems, to finally issue a conclusive opinion of the group's evaluation.

IV. The Mapping

In this section, it will be presented how the Software Testing Asset Mapping was developed [13].

Initially, as mentioned in the previous methodology, important competencies for the learning of Software Testing were identified. The authors selected six competencies, being: (1) Specify, design, implement, maintain and evaluate computer systems, using appropriate theories, practices and tools, (2) Apply methodologies that aim to guarantee quality criteria throughout all stages of the development of a computational solution, (3) Plan, specify, design, implement, test, verify and validate computing systems, (4) Understand and apply software development, evolution and evaluation processes, techniques and procedures, (5) Assess the quality of Software Systems, and (6) Conceive, apply and validate principles, standards and good practices in software development.

After the selection of the MEC competencies, its Mapping was carried out with several other assets to establish a knowledge base for the construction of software testing syllabus. The selected assets can be seen in Fig. 1.

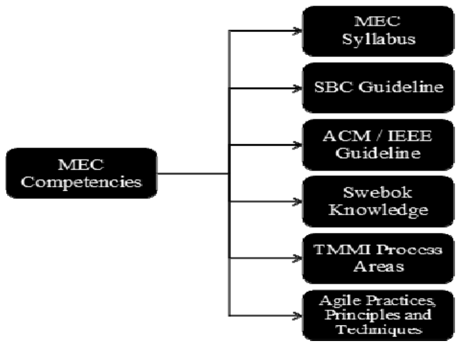

Fig 1 - Asset Mapping for the construction of Knowledge Units.

From this premise, for each of the highlighted assets, one of the MEC Competencies and its assets was correlated. Thus, the MEC Competencies helped to extract content from all the analyzed assets to assemble the proposal to build the teaching syllabus adhering to Software Testing.

The result extracted from the alignment of Fig. 1 served as a reference to guide the teaching units and areas that must be addressed in undergraduate courses in computing, however, they do not necessarily make it clear what the syllabus contents will be worked on, they only point out the minimum that must be taught. After this stage, a set of inputs was extracted from the alignment between the MEC competencie and each of the assets.

An input is an essential element to make a certain product or service. In this mapping, the set of results that were considered inputs were: output and input work products, and mainly content generated from each of the assets that were adhering to the highlighted MEC competencies.

As a result of this mapping, a total of 75 inputs were reached. Subsequently, the inputs were prioritized through a set of criteria selected by the researchers: Simplicity, Agile and Open Source Culture, Strategic Alignment and Benefit Comparison. After analyzing the criteria, one total of 40 inputs that helped in the construction of the syllabus.

V. THE SYLLABUS

The ACM / IEE Guideline [10] comments that there is no single formula for success in developing an information technology syllabus, although a group believes that the report's specific strategic recommendations and suggestions will be useful to a wide variety of institutions. This is important that the syllabus of the computing areas are always kept up to date, in order to adjust to the rapid changes in the field and with the teaching of computing in general in addition to meeting the needs of the software industry.

It is important that the development of syllabus for teaching and learning continues an ongoing basis, as with the constant changes in skills required by students in the labor market, the forms and content must be updated. In addition to considering perspectives for building students autonomy and their participation and protagonism in the current world, contributing to the students to stand out.

For the construction of a good syllabus, Harnish et al. [18] advise that they have eight learning components: (1) basic information about the course and contact information, (2) course purpose including goals and objectives, (3) instructor's teaching philosophy and beliefs, (4) assignments and course calendar, (5) required and optional materials including textbooks and supplemental readings such as journal papers, (6) methods of instruction and course delivery, (7) grading procedures, and (8) learning resources for students. This paper addresses the first three topics mentioned by [18], as the authors chose to separate the syllabus focused mainly on the construction of knowledge units and a teaching plan partially using this syllabus [19].

With the results extracted from the mapping, the construction of a syllabus model began, which, when fed with the acquired inputs, took place in different software testing teaching units using an agile context.

A. *Bloom's Taxonomy Revised*

For the construction of teaching units for this syllabus adhering to the teaching of software testing, the Bloom's Taxonomy Revised was adopted; it must guide the results through their forms of learning. Aiming to integrate new forms of educational developments, considering metacognition and constructivist theories that relate knowledge to the awareness of individual learning [20].

Taxonomy brought some changes in relation to the 1956 version [21], the main ones that we will present are in the Cognitive Dimensions [20] that have been updated to the new learning perspectives, being:

- Remembering: it is related to recognizing and reproducing learned ideas and content,
- Understanding: it is related to the establishment of a connection between the new and the previously acquired knowledge. The student can explain in his own words,
- Applying: lists the execution of a knowledge procedure in a specific or new situation,
- Analyzing: related to the understanding of relevant and irrelevant parts of certain knowledge and the understanding and correlation between different parts of knowledge,
- Evaluating: related to making judgments based on criteria and standards pertaining to the acquired knowledge,
- Creating: involves the development of new and original ideas, products and methods, using previously acquired knowledge and skills.

In the knowledge dimension of the bloom taxonomy [20], it is directly related to the content and aims to make the information about the content of the subject more meaningful for the student; it is divided into four types of knowledge:

- Effective / Factual Knowledge: involves the basic content that the student must master in order to achieve and solve problems;
- Conceptual Knowledge: related to the interrelation of basic elements in a more elaborate context where simple elements need to be connected for the formation of knowledge;
- Procedural Knowledge: involves knowledge of the achievement of an objective using methods, criteria, algorithms, and techniques. Abstract knowledge is stimulated;
- Meta-cognitive Knowledge: related to the recognition of cognition in general and awareness of the breadth and depth of knowledge acquired from a given content. There is a relationship with the knowledge previously assimilated to solve a certain problem.

Based on this knowledge, it was possible to build a proposal to create a syllabus model that later became teaching units adhering to software testing.

B. *Parameters for Syllabus Construction*

This research stage is essential for the creation of Teaching Units, as it seeks to establish the components for the construction of a generic syllabus, including: describing the objectives for that syllabus, defining the guiding questions and link the contents referring to the syllabus of each teaching unit, proposing problems, identifying the results and expected learning level, in addition to working on additional topics in each teaching unit.

The target of this syllabus is the teaching of software testing using an agile context as a framework to construction of program content. Table I shows the generic construction of the syllabus with the characteristics of each component.

The stage of defining teaching strategies is the determining factor for the elaboration of a syllabus, which is the planned way of teaching the topics of the study plan for each unit. A teaching strategy is chosen according to the level of learning intended for the topic and its expected results, at work [19] an approach to a teaching model that presents teaching strategies for tests in an agile context can be followed. For this research, this topic will not be addressed, as it is an instance of using the curriculum proposal.

TABLE I. GENERIC CONSTRUCTION OF A TEACHING UNIT

Teaching Unit	
Prerequisites	
These are the subjects, teaching units or axis that can facilitate learning if they are previously attended by students who must provide the foundation. It is advisable to indicate the reference syllabus that was used.	
Guiding Questions	
These are questions asked to students during the beginning of each unit, which aim to start discussing the topic.	
Programmatic Content (PC)	
This is the content that will be taught in the unit, in view of the competences planned for the teaching unit. Mapping was used to create topics for learning.	
Expected Results	***Level of Learning***
This is what the student must be able to learn and accomplish after the learning collected in the unit, always in an evolutionary character.	These are each of these expected results, which were detailed with a certain level of cognitive ability and knowledge dimension of Bloom's Taxonomy Revised.

C. *Teaching Units adhering to Software Testing*

The research delivers the construction of syllabus for a set of teaching units that researchers consider relevant for learning of testing in an agile context.

For each topic of each teaching unit, different activities must be performed and work products generated, so the proposed syllabus seeks to indicate content that can develop knowledge, skills and abilities aligned with software testing. The components chosen in all teaching units were based on the approaches mentioned above in conjunction with what was obtained in the literature review. Table II highlights the syllabus proposal for the first teaching unit.

Topic 1.1 in Table II was developed to provide the basic theoretical framework on testing and quality and the conceptualization of different testing terms, models and practices. The levels of learning together with the expected results reflect a more conceptual learning related to the search for knowledge and understanding of the classifications of concepts, rules and principles of testing and quality.

TABLE II. TEACHING UNIT OF SOFTWARE ENGINEERING

Teaching Unit 1 – Software Engineering	
Prerequisites	
ACM/IEEE: (SE) Software Processes SBC: Software Engineering	
Guiding Questions	
Q1. What testing practices guarantee quality software? (PC - 1.1, 1.2) Q2. How to find out which tests would be the most suitable for certain situations? (PC - 1.1) Q3. How to develop a testing strategy, which is suitable for the continuous evolution of software? (PC - 1.1, 1.2) Q4. What criteria should be considered for the development of software test scenarios? (PC - 1.1, 1.2,) Q5. What is the importance of agile practices, principles and techniques in the software testing? (PC - 1.1, 1.2)	
Programmatic Content (PC)	
1.1 Introduction to testing and quality 1.1.1. Conceptualization of testing and the agile testing 1.1.2. Testing fundamentals 1.1.3. Testing in the vision of agile methods 1.1.4. Software quality control 1.1.5. Levels of software testing 1.1.6. Types of software testing 1.1.7. Software testing techniques	
Expected Results	***Level of Learning***
The student must know the basic concepts of software engineering related to testing and quality control.	Remembering / Factual
The student must be able to see the relationship between test coverage and software quality.	Understanding / Procedural
Programmatic Content (PC)	
1.2 Introduction to software development and its development methods 1.2.1. Understanding of software. 1.2.2. Introduction to corrective software practices. 1.2.3. Fundamentals of agile practices in the software development process.	
Expected Results	***Level of Learning***
The student must understand the software development process.	Remembering / Procedrual
The student should be able to analyze and evaluate the quality of codes and be able to correct and understand it.	Analizing / Conceptual Evaluating / Factual

Topic 1.2 is an introductory step to understanding software and knowledge of different methods of software development, especially those related to agile practices. The level of learning to be achieved suggests teaching strategies for interpreting everyday situations in software development and the ability to compare and criticize rules and situations related to code development and quality practices.

Upon completion of the first teaching unit, the syllabus indicates a Software Development unit, shown in Table III, which according to SWEBOK is an area extremely related to the Testing areas.

Topic 2.1 in Table III was developed to provide concepts of identification of common errors, good coding practices, principles of code review and refactoring, aiming to keep the code readable. In addition to introducing students to the use of modern development aid tools. For this topic, it is important that the reflection of learning is recognition of terminology, details, and elements in addition to classifications, models and theories of software development.

In topic 2.2, the content focuses on programming techniques and good coding practices, already presenting the student with strategies for building testable codes and how to automate them and identify problems, defects and possible refactorings of the code. For this topic, it is indicated that the professor uses an object-oriented language; a modern development environment and that practical teaching activities are carried out. The levels of learning that must be achieved are understanding information related to programming techniques, analyzing problems of code development, maintenance or evolution, building solutions to these problems and being able to make judgments about codes and be able to indicate adjustments or even structural changes.

TABLE III. TEACHING UNIT OF SOFTWARE DEVELOPMENT

Teaching Unit 2 – Software Development	
Prerequisites	
ACM/IEEE: (SDF) Software Development Fundamentals SBC: Software Engineering	
Guiding Questions	
Q1. Is it possible that the entire functional code can be considerably evolutionary? (PC - 2.1) Q2. How do you identify that a code is becoming unreadable? How important is that? (PC -2.1) Q3. Is it necessary and important that a system already ready and working, keep having an update in your code? (PC - 2.1, 2.2) Q4. What criteria should be used for code reviews and refactoring? (PC - 2.2) Q5. What practices, techniques and tools facilitate: development, evolution and collaboration in systems? (PC - 2.1, 2.2)	
Programmatic Content (PC)	
2.1 Software development concepts 2.1.1. Basic concepts 2.1.2. Identification of common coding errors 2.1.3. Principles of code review and refactoring 2.1.4. Importance of good practices and keeping the code readable 2.1.5. Identification and understanding of modern programming environments (IDE)	
Expected Results	***Level of Learning***
The student must know the basic concepts of software development.	Remembering / Factual
The student must be able to see the relationship between code development and its evolution.	Remembering / Conceptual
Programmatic Content (PC)	
2.2 Defining and implementing defensive programming and software maintainability techniques 2.2.1. Code analysis and relationship with good practices 2.2.2. Conducting a review and analysis of the possibility of refactoring 2.2.3. Strategy definition and use of unit tests 2.2.4. Perform code automation and integration	
Expected Results	***Level of Learning***
The student must know the basic concepts and work products related to the software development and its use in relation to testing.	Remembering / Factual
The student must be able to know and execute strategies and good practices for software evolution and maintenance.	Remembering / Factual Applying / Procedural
The student must be able to propose adjustments and improvements to the delivered code.	Eveluating / Conceptual Creating / Meta-cognitive

The next teaching unit that the syllabus aims to be created is that of Quality, shown in Tabela IV.

TABLE IV. TEACHING UNIT OF QUALITY

Teaching Unit 3 – Quality	
Prerequisites	
ACM/IEEE: (SE) Software Design SBC: Software Engineering	
Guiding Questions	
Q1. What is software design? Is there more than one way to design software? (PC - 3.1) Q2. Are Design Patterns Important for Software Development? Do they facilitate use and coupling with tests? (PC - 3.1) Q3. What ways are there to build requirements? Is it possible to write tests in conjunction with requirements? (PC - 3.1, 3.2) Q4. Which criteria, practices or principles are important to assess quality of systems? (PC - 3.1, 3.2)	
Programmatic Content (PC)	
3.1 Software quality concepts 3.1.1. Principles of system design 3.1.2. Structural and behavioral models of software designs 3.1.3. Design Patterns 3.1.4. Requirements Design	
Expected Results	***Level of Learning***
The student must know the basic concepts of software quality.	Remembering / Conceptual
Programmatic Content (PC)	
3.2 Software quality strategies aligned to testing 3.2.1. Use design components to measure quality 3.2.2. Understand refactoring using design patterns 3.2.3. Elucidation of software design through requirements and tests 3.2.4. Construction of requirements work products using testing principles and agile methods.	
Expected Results	***Level of Learning***
The student must understand the use of design patterns in the software development.	Remembering / Factual Understanding / Conceptual
The student must be able to elucidate and build work products using design patterns with testing strategies and agile methods.	Analizing / Factual Applying / Procedural e Meta-cognitve

Topic 3.1 in Table IV was developed to reinforce the basic concepts of software quality related to testing and development patterns. The intended learning is that students can solve problems of tasks that require several elements.

In topic 3.2, the content to be taught is that of strategies for assessing code quality, mainly using software testing, in addition to understanding the importance that tests are present in all stages of development, from requirements. Regarding the learning of this topic, the student must know and learn how to use code design patterns in the development stages, which were presented in Unit 2. Therefore, the student must be able to build testable and agile wor products and understand the importance of their uses and know how to apply them in projects. For this topic, the use of practical teaching activities in an application instance is indicated. The authors would advise the use of test-oriented requirements construction techniques for this Teaching Unit.

The last teaching unit of the syllabus is the most important, that of software testing, shown in Tabel V, which will consolidate the knowledge acquired in the previous units. This teaching unit, when put into practice, should be used with the maximum of practical teaching strategies, and was strongly influenced by the TMMi model.

In topic 4.1 of table V, as in other teaching units, knowledge of test concepts, agile methods, quality, among others, was built. One of the objectives of this unit is to end the test learning cycle by integrating previously acquired knowledge. In this the student must build software test work products using tools to support the construction of them. Learning in this topic comes in the form of the student knowing the techniques for creating test work products, analyzing whether these work products will evolutionarily support a software project. It is advisable those practical teaching sessions in teams are used throughout this unit and that professors use some practice that continues to simulate a real software development or maintenance project.

TABLE V. TEACHING UNIT OF SOFTWARE TESTING

Teaching Unit 4 – Software Testing	
Pré-requisitos	
ACM/IEEE: (SE) Software Verification and Validation SBC: Software Engineering	
Guiding Questions	
Q1. Is it possible to develop software being oriented towards its testing? (PC - 4.1, 4.2, 4.3, 4.4) Q2. How should we start a Testing project? Are there different practices, tools and techniques for this? (PC - 4.1) Q3. How should we identify non-testing software problems? (PC - 4.2) Q4. Monitor Requirements and Testing, are they necessary in software development? How changes in requirements and scope should be handled (PC - 4.2, 4.3) Q5. Is it possible to carry out a test-driven development project that can always be used? (PC - 4.4)	
Programmatic Content (PC)	
4.1 Initial concepts for developing testing approaches 4.1.1. Test plan creation 4.1.2. Construction of test cases 4.1.3. Approaches to verification 4.1.4. Types of testing tools and their role in software validation 4.1.5. Performance metrics and testing coverage	
Expected Results	***Level of Learning***
The student must know the basic concepts, tools and testing approaches and must know how to use them.	Remembering / Factual Understanding / Procedural
The student should be able to see the relationship between testing concepts and their role in software evaluation.	Understanding / Procedural Analyzing / Conceptual
The student must be able to build software testing work products.	Creating / Procedural
Programmatic Content (PC)	
4.2 Construction of work products for validation and verification of software testing 4.2.1 Defect identification and tracking 4.2.2. Methods of creating test-oriented requirements 4.2.3. Test-oriented software development methods 4.2.4. Verification and validation of non-code related work products 4.2.5. Estimation of time and risk involving testing	
Expected Results	***Level of Learning***
The student must be able to identify and understand development methods guided by testing.	Remembering / Factual
The student must be able to know and understand verification, validation and risks of software testing.	Understanding / Factual Evaluating / Conceptual
Programmatic Content (PC)	
4.3 Test-oriented development project 4.3.1 Software evaluation process through requirements and testing 4.3.2 Prototype construction approaches and requirements. 4.3.3. Execution of test-oriented development using good practices and design patterns 4.3.5. Develop integration, automation and evolution of code and testsing 4.3.5 Approaches and decision making related to software development	
Expected Results	***Level of Learning***
The student must be able to know and perform activities related to software development and testing.	Understanding / Factual
The student must be able to perform and evaluate development tasks, changes in development and software testing.	Evaluating / Conceptual Applying / Procedural e Meta-cognitive
Programmatic Content (PC)	
4.4 Evaluation of test-oriented project 4.4.1. Verification between requirements and developed project 4.4.2. Obtaining coverage and quality of testing measured 4.4.3. Improvements in the testing process	
Expected Results	***Level of Learning***
The student must be able to propose adjustments and improvements to testing process models.	Evaluating / Factual Applying / Conceptual

In topic 4.2 the content tries to consolidate the student's knowledge about the construction of software verification and validation work products, where the student must learn about defect identification and tracking, work product construction methods guided by testing, verification and validation of work products and estimating development time and risk with testing coverage. Student learning is aimed at recognizing and solving problems about work products and being able to interpret and verify situations of test work products connections.

Topic 4.3 presents content focused on how to build software through testing requirements and work products, build testable prototypes, learn how to develop software guided by testing practices and good code building practices. Also, the decision-making process for software development situations, such as those agile methods accept and act when changes are necessary. Learning will take place in the form of the student being able to understand and solve problems, verify work products, develop solutions, and implement actions with the knowledge assimilated in conjunction with his / her previous skills.

Topic 4.4 concludes the teaching unit, where the contents presented are about evaluation of work product development, quality analysis and improvements in software processes. Learning takes place in a way that students develop criticality about software products and that through the acquired knowledge they can propose suggestions and improvements to both the developed work products and the taught process.

Finally, the set of techniques and practices adopted to compose the activities of the teaching units, in addition to being based on literature review and mapping, was also based on training of professionals in the software testing industry [22] for the construction of the programmatic content.

VI. The Evaluation

As previously described, an evaluation of the proposed study plan, which adheres to the teaching of testing in an agile context, took place in a panel of experts who must follow a set of requirements, such as: having teaching and professional experience in the area of Software Engineering, hold a doctoral degree, research publications related to the teaching of some area of computing, having participated in the creation or updating of your institution's pedagogical project, and having adopted some practical teaching approach.

Thus, three doctoral professors at Brazilian federal public universities, who have the requirements described above, were selected for the panel of experts. This panel evaluated the criteria defined in Table VI, related to the documentation and usability of the curriculum, issuing a conclusive opinion at the end.

TABLE VI. Results of the Evaluation by Expert Panel

Category	Criteria	Expert Opinion
Syllabus Documentation	Adequacy	Yes
	Clarity	Yes
	Completeness	Yes
	Changeable	No
	Detailing	Yes
Syllabus Usability	Ease of use	Partially agree
	Flexibility	Disagree
	Effort Involved	Moderate Effort
	Skills Required	Moderate Skills

From the analysis of this evaluation and the opinion of the consensus meeting, it is concluded that the experts' panel considered the syllabus adequate, complete and detailed for teaching of software testing. The experts point out that as the syllabus has not yet been applied and lacks an example of use to be analyzed, it ended up making it difficult to assess its usability. According to the opinion of an expert: "I believe that a professor with moderate knowledge about testing and tools in the field can apply" and another expert pointed out: "The model is very prescriptive and leaves no scope for adaptation", which was considered when was introduced to the generic construction of a teaching unit, so that other professors can replicate the construction of teaching units in the syllabus.

On strengths that the experts found in applying the syllabus collected during the consensus meeting: “The syllabus is well-founded and structured. Its activities and recommendations are based on published papers, which validates the prepositions”, and “It considers important to create devices for teaching of software testing in a practical way. Just as it is interesting to approach the agile context, since it is widely used in the industry”.

As recommendations, the experts cited that an instance of application of the syllabus should be developed; aiming to put into practice what was built in a real teaching situation. This instance must propose teaching strategies, being guided by the guiding questions and programmatic content, and must achieve the expected results.

VII. Conclusion

The goal of this research was to investigate and contribute to a software testing teaching syllabus, consisting of a set of teaching units developed with the perspective of supporting students' learning and preparing them for the professional market. On the research question, the syllabus developed from reference guides, quality models and agile practices used in the software industry in conjunction with the developed syllabus construction, generated teaching units that had a favorable opinion on the documentation and an average opinion on usability by the experts panel.

The syllabus was developed from a mapping and supported by a literature review, based on existing studies, to classify content, results, levels of learning and knowledge. The authors suggest that during the execution of the syllabus, as many practical activities as possible are given, given that academic syllabus for certain areas, such as software development, which involve a variety of activities and the conventional form of syllabus approach, end up not resembling what happens in the software industry [7].

Using teaching units from different knowledge areas, but all related to software testing, we sought to create an integration of content in an evolutionary way with knowledge to contribute to student learning.

The main contributions of this work are: the approach to syllabus construction, which can be replicated by other authors considering the same theme or similar topics in the computing, since the reference guides used are compatible with most computing courses, the teaching units that were developed for teaching of software testing with syllabus content enriched by industry practices and agile principles, practices and techniques. We can also consider that other researchers to create teaching units, subjects, or syllabus can use the generic syllabus for the creation of a teaching unit.

There are limitations in this research presented, perhaps the main one being the lack of an instance of syllabus application. Only then it will be possible to identify the improvements to be included in this syllabus and even forms of application with different tools, techniques, and programming languages. Another limitation would be the lack of a repository for storing experiences using instances of use of the syllabus, this can benefit professor who want to use the syllabus to find available materials that facilitate the learning curve for their use.

The threat to external validity is a condition that limits the ability to generalize the results of a research, for this research an external threat was that a part of the assets used in mapping to construct the syllabus were from reference guides and Brazilian teaching guidelines. Thus, the generalization of results is limited to other places with very different syllabus guidelines and reference guides.

The main future work in progress is the creation of an instance of using the syllabus, which is a teaching plan on how to apply the syllabus, presenting the teaching stages that can be used, support materials that can stimulate learning, as well as active learning methods such as: playful teaching and gamification.

Acknowledgements

The authors would like to thank CAPES (Coordination and Improvement of Higher Level or Education Personnel) for granting an institutional Doctoral scholarship to the author of this work and the experts who participated in the panel evaluating the research proposed in this paper.

References

[1] R. S. Wazlawick, "Software Engineering Concepts and Practices". 2. ed. Rio de Janeiro. Elsevier, 2019

[2] L. Y. Tao, "Research and Application of an Agile Software Testing Model based on Scrum". master's thesis of Huazhong University of Science and Technology, 2011.

[3] ACM/IEEE, "Computer science curricula 2013". Curriculum guidelines for undergraduate degree programs in Computer Science. 2013.

[4] SBC - Sociedade Brasileira de Computação, "SBC Reference Curriculum for Undergraduate Courses in Bachelor of Computer Science and Computer Engineering". Grupo de trabalho responsável – CR2005. 2005.

[5] P. Bourque and R. E. Fairley, "SWEBOK Guide V3.0". 2014. Available: www.swebok.org.

[6] R. S. Pressman and B. R. Maxim, "Software Engineering; A Practitioner's Approach", McGraw-Hill, 2015.

[7] A. M. R. Vincenzi, M. E. Delamaro, A. C. D.Neto, S.C.P.F. Fabbri, M. Jino, and J. C. Maldonado, "Automating software testing with open source tools". Rio de Janeiro. Elsevier, 2018.

[8] MEC, "National Curriculum Guidelines for Undergraduate Computing Courses (DCN16)". Brazil. 2016.

[9] F.A. Zorzo, D. Nunes, E. Matos, I. Steinmacher, J. Leite, R. M. Araujo, R. Correia, and S. Martins, "Training References for Undergraduate Computing Courses". Sociedade Brasileira de Computação (SBC). 153p, 2017. ISBN 978-85-7669-424-3

[10] B. V. M. Sabin, H.Alrumaih, J. Impagliazzo, B Lunt, M Zhang, B Byers, W Newhouse, B Paterson, S Peltsverger, and C Tang, "Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology". A Report in the Computing Curricula Series Task Group on Information Technology Curricula. ACM, 2017.

[11] TMMI Foundation, "Test Maturity Model Integration – TMMi Release 1.0". 2018.

[12] S. Laing and K. Greaves. "The Testing Manifesto". 2015. Available: http://www.growingagile.co.za/2015/04/the-testing-manifesto.

[13] I. Elgrably and S. Oliveira, "A Proposal for Teaching and Applying Focused of Agile Tests Methods made by an Assets Mapping". 16th International Conference on Information Systems & Techology Management – CONTECSI. 2019.

[14] L. P. Scatalon, M. L. Fioravanti, J. M. Prates, R. E. Garcia, and E. F. Barbosa, "A survey on graduates' curriculum-based knowledge gaps in software testing". in 2018 IEEE Frontiers in Education Conference (FIE), San Jose, 2018.

[15] O. A. L. Lemos, F. F. Silveira, F. C. Ferrari, and A. Garcia, "The impact of software testing education on code reliability: an empirical assessment". Journal of Systems and Software. 2017.

[16] P. H. D. Valle, E. F. Barbosa, and J. C. Maldonado, "Cs curricula of the most relevant universities in brazil and abroad: perspective of software testing education". International Symposium on Computers in Education (SIIE), 62–68. 2015.

[17] E. Bjarnason, M. Unterkalmsteiner, M. Borg, and E. Engström, "A Multi-Case Study of Agile Requirements Engineering and the Use of Test Cases as Requirements". in: Information and Software Technology, vol.77, pp. 61-79. 2016.

[18] R. J. Harnish, R. O'Brien McElwee, J. M. Slattery, S. Frantz, M. R. Haney, C. M.Shore, and J. Penley, "Creating the foundation for a warm classroom climate: Best practices in syllabus tone". APS Observer, 24, 23–27. 2011.

[19] I. Elgrably and S. Oliveira, "Model for teaching and training software testing in an agile context". 50th Annual Frontiers in Education – FIE'20. Uppsala – Sweden. 2020., in press.

[20] A. P. Ferraz and R.V. Belhot, "Bloom's taxonomy: theoretical review and presentation of the instrument's adjustments to define instructional objectives". Gest. Prod., São Carlos, v. 17, n. 2, p. 421-431, 2010.

[21] B. S. Bloom, "Taxonomy of educational objectives". Book 1: Cognitive domain. New York: David Mckay. 1956.

[22] ISTQB, "International software testing qualifications board". 2018, Available at: https://www.istqb.org/.