Model for teaching and training software testing in an agile context

Isaac Souza Elgrably
Graduate Program in Computer Scienes (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 Scienes (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 a proposal for a model to improve the teaching process of software testing supported by elements of agile methods. It was developed to be used as a guide to support professors or specialists for decision making, selection of teaching materials according to the participants' cognitive learning for the execution of subjects or teaching units on topics related to software testing. In addition to suggesting several personalized activities, presenting an instance of application of a teaching syllabus that adheres to several reference syllabus, this model also seeks to be extensible, to be applied in other teaching areas, trying not to be limited to the software testing area. The construction of this model makes use of practical teaching approaches supported by practices, such as gamification and playful teaching to motivate and engage participants, bringing common tasks and technologies in the software industry in parallel to classic lecture classes, in order to develop certain skills, which were obtained through academic syllabus, together with technical skills in software testing for participants, in addition to knowledge of several open source tools that are used by testing professionals in the software industry. This construction is specified in several stages arranged in a teaching cycle, being sequenced when they are applied and can be adapted for other learning scenarios. Subsequently, the skills, competences and support materials that can be used are presented in addition to the possible effects expected on students from the application of this model in the teaching of software testing. As it uses practical teaching approaches, it is advisable that this model is used by professors who have a teaching facilitator profile, as it presents humanistic teaching learning, with a focus on participants and teamwork, principles found in agile methods, in which activities will have deliveries of products that approach what is performed in a software company and it will be necessary the interaction of all participants in the model. The evaluation process for this teaching model took place in two stages. First, there was an analysis and individual evaluation of the model through the questionnaire, then there was a consensus meeting between the evaluators, where they justified the marks attributed in the evaluation and their considerations on other responses. This meeting aimed to solve disagreements and collect opinions, enabling problems to be solved and reaching a final consensus among the evaluators. The authors consider that it is necessary to remodel teaching paradigms of computing subjects, using together some characteristics of traditional approaches with more practical teaching approaches that solidify the knowledge of the participants for the challenges that exist in the software industry.**

Keywords — software testing, teaching model, focus on student, technical skills

I. Introduction

The software testing activity aims to guarantee software quality, inherent to several negotiated requirements and subsequently established between customers and development teams, these activities must follow functional requirements, specifications explicitly described and an established performance [1]. As an important activity to guarantee quality, the software industry performs software testing activities in a systematic way, since if the software is not tested, its quality may be compromised [2].

Considering the importance of the software testing activity, this research proposes a model that makes use of practical approaches and active teaching methodologies. Bringing common tasks and technologies in the software industry as part of a testing teaching plan, aiming to develop technical skills in software testing for participants.

As an important task for the software industry, Benniti [3] saw the need to support the teaching of software testing in a flexible way to adapt to different teaching contexts (training and undergraduate courses). Additionally, there was an observation that undergraduate courses in computing area, in general, do not provide students with an integrated view of software test contents with other undergraduate courses [4].

Thus, this model is composed of a software testing teaching syllabus adapted from [5], which was built based on a set of reference guides for computing courses [6][7][8][9] and aligned with a set academic competences of MEC (Ministry of Education in Brazil) [10] and a practical teaching plan focused on the participant to try to deliver an integrated and evolutionary test knowledge with different techniques and practices.

Researches show that there is a certain deficiency in the teaching of some computing topics that may have a relationship with the way in which professors teach computer content, as there are still few initiatives that use teaching methodologies other than the traditional one [11]. Efforts should be encouraged to adopt pedagogical models that aim to promote scenarios that provide active learning to software test students [12].

Thus, it is advisable that this model is used by professors who have a teaching facilitator profile, as it presents humanistic teaching learning, with a focus on the participant and teamwork, in which the activities will have deliveries of products that approach the which is carried out in a software company and the interaction of all participants in the model will be necessary.

Another choice for the teaching plan was to use practices and techniques derived from agile methods, because in its testing approach, a greater focus on human activity is used than in the generation of documents, with the individual being the center of the process. What aligns perfectly with what it is defended by active teaching methodologies.

Some of the conclusions that this research seeks, can be seen in these questions:

- Research Question I (RQ I): *Is it possible to build a test teaching plan, using active methodologies, focused on practical tasks and agile practices that are adequate to guarantee learning related to software tests?*
- Research Question II (RQ II): *Can a software teaching and training model using agile context be adherent to a test teaching syllabus based on academic competencies?*

To answer the RQ I, it is considered the need to adjust the academic teaching methods aimed at testing [11][13]. For this, the authors adapted a syllabus proposed by [5] and developed a teaching plan for the transfer of knowledge of the program content. Subsequently, the results were presented to a group of doctors in software engineering who are also researchers in teaching software engineering.

For resolution of RQ II, a set of academic skills that the authors considered important to assist in the generation of skills necessary in job market was highlighted. Later, the teaching plans created from the syllabus must be able to translate this knowledge to the participants and the panel of experts also evaluated the results on that.

In short, the goal of this model is to assist in the teaching of participants, through the creation of a teaching plan that will be an instance of using a syllabus built for testing teaching, aiming to materialize the learning of software testing skills through closer activities of those that will be found in labor market using approaches that facilitate the practical and cognitive learning of the activities presented.

In addition to this introductory section, this paper is structured as follows: in Section II the theoretical basis is provided on the test model and its main active methods, in Section III the research methodology is described, presenting its organization and what results must be achieved, in Section IV a test teaching syllabus for the creation of the teaching plan is presented, in Section V a teaching plan showing the use of the model will be presented, in Section VI the evaluation of the model is presented by means a expert panel, and, finally, in Section VII we concluded this work, presenting the contributions of this model, in addition to presenting its limitations and possible threats to its validity and highlighting our plans for future studies.

II. Background

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

A. *Teaching of Testing*

In the literature there is already a line of research that addresses the use of different teaching strategies to make the student as central target in the learning process [2][3][4][11]. A group of researchers argues that there is need for a change in teaching strategies of software testing through changes in teaching approaches and / or pedagogical models that promote the student in the learning center, leaving the responsibility for learning to the student [11].

Another need is that students materialize the learning of software tests through activities closer to those that will be found in labor market. Thus, one of the segments we can address is that to build new knowledge, especially those sought in the job market, the teaching and learning process is crucial, with a more student-centered approach to teaching, the teacher is no longer a content transmitter and starts to guide students, mediating and stimulating learning [14].

One of the difficulties in the software testing teaching process is that it ends up encompassing several different activities, such as: planning, analysis, modeling, among others. So, it ends up becoming a challenge to build a student's education prepared to try to meet the expectations of the job market. In Valle's research [4], it was observed that undergraduate courses in the field of Computing, in general, not provide students with an integrated view of software testing content with other undergraduate courses. Thus, without this integration of different software testing content, it makes it difficult for students to prepare for the job market.

Some different initiatives can be used to train and improve the skills of software testing professionals, which are: educational modules, peer testing, TDD (Test Driven Development), software testing teaching with programming, educational games, among others [15] . They served as an aid to the construction of the teaching plan.

Thus, we found that it is essential that this presented model considers problems already observed in the literature, however, in addition to the active learning models, it is necessary to promote greater engagement of the participants.

B. *Active Teaching Methodologies*

Active teaching methodologies aim to encourage the academy to propose teaching models that develop autonomous and participatory learning capacity by students. Accordingly, MEC [10] proposes that the teaching methodology for computer courses be centered on the student as an object of learning and supported by the teacher as a facilitator of the teaching-learning process.

An active teaching methodology is a strategy that places a participant in the central role of learning, being the protagonist and the teacher a mediator and an auxiliary of the teaching-learning process.

In active learning methodologies, the involvement of participants is sought through activities or discussions, aiming at the exchange of knowledge, and emphasizing the experiences between all, usually with group work [16].

Therefore, this teaching plan was developed thinking about the benefits of active methodologies, among them: generating learning through the resolution of practical problems, students as protagonists in learning, enabling the student to develop different skills in the teaching plan, among others. Some forms of active teaching methodologies used to build this teaching plan were:

1) Gamification: Gamification can be used so that students' live experiences of learning as a teaching methodology uses game elements for these purposes.

Software engineering as a whole faces challenges with the training and preparation of future professionals to improve practices and techniques [17]. Therefore, the use of teaching strategies using gamification resources are already used to improve the aid and educational engagement in teaching software testing as a motivating agent [17][18].

This teaching plan adopted gamification as an active methodology resource to motivate students to participate and engage in the learning process, to engage in the proposed challenges, and it also works as feedback for students to follow their development throughout during the activities.

2) Dojo: As an active methodology for preliminary coding teaching, the coding Dojo technique is very promising, in addition to being widely used to teach students in materials related to software engineering [19].

The practice of programming in pairs together with dojos are practices increasingly adopted for the development of cooperative software [16], especially in teams that use agile methods.

In the way that this teaching plan wants to make the student the center of learning, there was an opportunity to use Dojos so that students could transfer previous experiences and knowledge among themselves.

3) Playful Teaching: Recently, has been an increase in the use of ludic approaches for teaching and learning in undergraduate education [20], one of the reasons for this apparent panorama is that the ludic function aims to amuse the student by providing opportunities for their learning.

They are mainly used for the creation of educational games, however, they can be used even for a syllabus redesign to try to provide learning [21], using tools or techniques that should promote a search or development of knowledge [22].

The adopted teaching plan uses this active methodology to try to facilitate learning and create fun activities for students.

C. Related Works

Based on the premise of this research, to create a teaching plan for a comprehensive teaching of different testing practices, a search was made in the specialized literature with an emphasis on published articles.

Several practical works developed on teaching software tests are games related to the teaching of some technique or type of test [2][14][17][18]. This work is different because it presents a conceptual model for software testing teaching, presenting a syllabus with programmatic content tests, the academic skills that are the objectives to be achieved by the participants, and a teaching plan using active teaching methodologies and focused on agile practices.

There are works that approach the teaching of tests more broadly, in which the authors use learning objects [3] or games [11] to teach several topics of software tests. The work presented differs from these in that it prioritizes practical activities and tasks like those found in the software industry to be used to personify the programmatic content presented.

Finally, there are still works that present test teaching models [23] that present the use of a teaching strategy in conjunction with several learning objects to provide software testing teaching in a practical way. The main differences is that in this research, the proposed model presents its own syllabus guided by the learning objectives of Bloom's Revised Taxonomy [24], which collaborates significantly, as it is an instrument for classifying learning objectives hierarchically (evolutionarily) used for planning different teaching objects.

III. Research Methodology

To achieve the objectives of this paper, a syllabus adhering to software tests was used [5]. Then a subject was created for the application of this model and an instance of application was created using a set of active teaching methodologies, to become a teaching plan. Fig. 1 shows the methodology applied in this research.

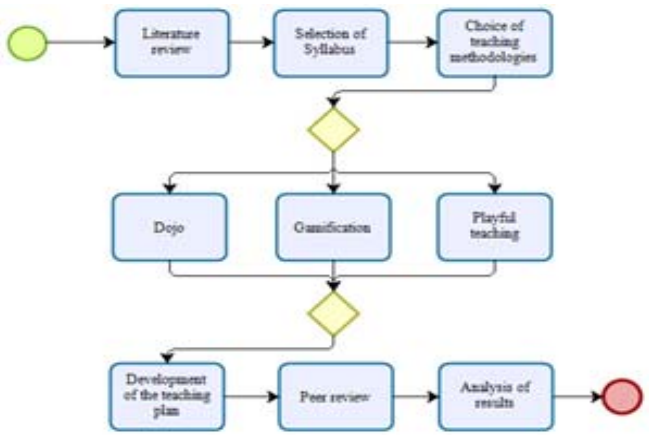


Fig. 1. Methodology that was applied in the research.

Initially, a literature review was carried out on the problems encountered in the area of teaching software testing, the reference guide [9] indicates that in the area of computer teaching subjects should seek to be oriented towards practical activities and also states that the part practice of applying information technology is essential for career success. In addition to the research by Paschoal and Souza [23] that most professors use practical work in the subject because software testing is a very practical content, similar to software coding.

Based on the above premises, the authors adapted a syllabus [5] that has the program specifically aimed at sofware testing teaching. After analyzing the syllabus content, active teaching methodologies that worked together with the perspective of testing teaching found by the syllabus were designed. Then, through the review of what was collected in [3][4][15][24] and practical teaching experiences obtained by the authors [25], the use of Dojo, gamification and playful teaching methodologies were selected to motivate and strengthen learning.

Subsequently, a teaching plan was built in which teaching strategies complement the programmatic content that will be taught, always seeking to adopt a humanistic approach to teaching in conjunction with agile methodologies, open source tools or learning objects that deliver practical knowledge to participants and that aligned with what is needed in labor market.

The evaluation of the model was carried out in two stages. Initially, a trio of experts in teaching software engineering, by means of an electronic questionnaire, evaluated the content presented in a preliminary version of the model. Subsequently, there was a consensus meeting between the authors and experts with the aim that the problems encountered were discussed, ideas were explained and collected opinions allowing adjustments to be made to the preliminary version of the model.

IV. The Syllabus

The goal of this model is to support the teaching of Software Tests in undergraduate courses in the Computing area.To present the test teaching model of this research, it is necessary to use a syllabus to create the teaching plan. Thus, it was decided to adopt the syllabus proposed in [5], since it originated from an asset mapping with a focus on agile methods [26]. This syllabus follows as a reference a set of competencies and syllabus guidelines from MEC [10], which is the regulatory body for syllabus guidelines for bachelor's and undergraduate degrees in computing in Brazil. These guidelines serve as a foundation for the realization of courses in the Information Technology areas, 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.

As a starting point for the creation of this syllabus, an alignment was made between the MEC competencies that the authors determined to be essential for teaching software tests, identified in Table I.

TABLE I. LIST OF THE MEC COMPETENCIES ADOPTED

MEC Competencies	
Comp1	Specify, design, implement, maintain and evaluate computer systems, using appropriate theories, practices and tools.
Comp2	Apply methodologies that aim to guarantee quality criteria throughout all stages of the development of a computational solution.
Comp3	Plan, specify, design, implement, test, verify and validate computing systems.
Comp4	Understand and apply processes, techniques and procedures for building, evolving and evaluating software.
Comp5	Assess the quality of software systems.
Comp6	Conceive, apply and validate principles, standards and good practices in software development.

The participants must acquire these skills at the end of a teaching cycle using the model presented.

The syllabus that will be used in this research, adapted from [5], has the following structural characteristics and can be seen in full in Table II:

- Prerequisites: are the subjects that can facilitate learning if they were previously attended by the participants, were considered subjects in the syllabus [7][8],
- Guiding Questions: these are questions asked to the participants during the beginning of the learning process,
- Programmatic Content: this is the content that will be taught in each teaching topic. The definition of these contents was also guided by the mapping of inputs carried out in [26],
- Teaching Strategy: this is the planned way to teach the topics of the syllabus for each unit. The teaching strategy is defined according to the level of learning intended for the topic and its expected results,
- Expected Results: the learning is acquired, they must be evolutionary and enable the participant to achieve the training of the skills planned for each of the competencies,
- Level of Learning: each of these expected results, which were detailed with a certain level of expected cognitive ability, using a terminology based on Bloom's Taxonomy Revised [27].

TABLE II. SYLLABUS OF SOFTWARE TESTING SUBJECT

Subject – software testing	
Prerequisites	
ACM/IEEE: Software Engineering SBC: Software Engineering	
Guiding Questions	
Q1. What is the importance of agile practices, principles and techniques in the universe of software testing? (PC - 1.1, 1.2) Q2. How do you identify that a code is becoming unreadable? How important is that? (PC -1.2) Q3. What practices, techniques and tools facilitate: implementation, evolution and collaboration in systems? (PC - 1.1, 1.2,1.3,1.4) Q4. What ways are there to build requirements? Is it possible to write tests in conjunction with requirements? (PC - 1.3, 1.4) Q5. What criteria, practices or principles are important to assess quality of systems? (PC - 1.3, 1.4) Q6. Is it possible to develop software being oriented towards its tests? (PC - 1.2, 1.3, 1.4) Q7. Monitor Requirements and Tests, are they necessary in software construction? How changes in requirements and scope should be handled (PC - 1.3, 1.4) Q8. Is it possible to do a test-driven development project? (PC - 1.4)	
Programmatic Content (PC)	
1.1 Introduction to software testing 1.1.1. Conceptualization of tests and the agile testing context 1.1.2. Testing fundamentals 1.1.3. Testing in the vision of agile methods 1.1.4. Levels, types and techniques of software testing	
Expected Results	***Level of Learning***
The participant must know the basic concepts of software engineering related to testing and quality control.	Remembering / Factual
The participant must be able to see the relationship between testing coverage and software quality.	Remembering / Procedural
The participant must be able to analyze and evaluate the quality of codes and be able to correct and understand it.	Analyzing / Conceptual Evaluating / Factual
Programmatic Content (PC)	
1.2 Software Development Concepts 1.2.1. Introduction to corrective software practices 1.2.2. Fundamentals of agile practices in the software development process 1.2.3. Principles of code review and refactoring 1.2.4. Importance of good practices and keeping the code readable 1.2.5. Identification and understanding of modern programming environments (IDE)	
Expected Results	***Level of Learning***
The participant must know the basic concepts of software development.	Remembering / Factual
The participant must be able to see the relationship between code development and its evolution.	Remembering / Conceptual
The participant must be able to know and run strategies and good practices for software evolution and maintenance.	Remembering / Factual Applying / Procedural
The participant must be able to propose adjustments and improvements to the delivered code.	Evaluating / Conceptual Creating / Meta-Cognitive
Programmatic Content (PC)	
1.3 Software Quality Strategies in line with Testing 1.3.1. Structural and behavioral models of software projects	

1.3.2. Elucidation of software design through requirements and testing 1.3.3. Construction of requirements work products using testing principles and agile methods	
Expected Results	***Level of Learning***
The participant must understand the use of design patterns in software development.	Remembering / Factual Understanding / Conceptual
The participant must be able to elucidate and build wor products using design patterns with testing strategies and agile methods.	Analyzing / Factual Applying / Procedural and Meta-Cognitive
Programmatic Content (PC)	
1.4 Test-oriented development project 1.4.1. Methods of creating test-oriented requirements 1.4.2. Test-oriented software development methods 1.4.3. Improvements in the testing process 1.4.4. Software evaluation process through requirements and testing 1.4.5. Verification between requirements and developed project 1.4.6. Obtaining coverage and quality of testing measured	
Expected Results	***Level of Learning***
The participant must be able to identify and understand test-driven development methods.	Remembering / Factual
The participant must be able to know and perform activities related to software development and testing.	Remembering / Factual
The participant must be able to perform and evaluate development tasks, changes in software development and testing.	Evaluating / Conceptual Applying / Procedural and Meta-Cognitive
The participant must be able to propose adjustments and improvements to testing process models.	Evaluating / Factual Applying / Conceptual

In the teaching plan presented in the next section, there will be the construction of the teaching strategy for each syllabus presented to achieve the expected results. The model will use active teaching methodologies in conjunction with teaching tools, practices and techniques of agile methods used in industry and also activities that can be used as an example of use will be shown.

V. The Teaching Plan

In order to fulfill the goal of this model, it is necessary to create a teaching plan focused on the student, with the professor as a supportive figure and the knowledge generated must approach what is required in labor market. It should try to break the traditional teaching paradigm based on the theory that the professor only disseminates knowledge via classical lecture class.

The developed teaching environments aim to determine a better way to teach a subject to its participants and should encourage critical thinking and seek to provide the highest degree of learning possible for the participants. A gamification based on the ideas adopted by Chou and his Octalysis Framework [28] will be used to provide engagement to the participants, their monitoring will be done via a shared electronic spreadsheet, in which a judge will collect the results of the participations that occurred. Table III shows the participants' scoring methods.

TABLE III. Use of Gamification in the Model

Class Type	Games Elements	Points in the Spreadsheet
Classical lecture class	Points, ranking, narrative and real-time control	Presence, participation and suggestions
Dojo Class	Points, ranking, learning curve, step-by-step tutorial, group activity	Presence, participation, suggestions, initiative and participation in Dojo
Challenge list	Points, ranking and learning curve	Grade from the challenge list
Evaluative activity	Points, ranking and challenges list	Test grade and presence

Then, the teaching plans for each of the four syllabuses of the software testing subject will be highlighted.

A. Introduction to software testing

This first teaching topic was developed to give the basic theoretical framework on testing and quality and the formulation of different terms, models and testing practices.

This content will be taught through classic lecture classes supported by the parallel reading of scientific papers and a manual on testing practices, demonstration of testing practices being presented and video lessons provided to participants in parallel to practical classes in order to learn and execute the indicated concepts.

The conceptual material selected for these classic lectures will be heavily based on the International software testing Qualifications Board (ISTQB) guides: Foundation Level (*https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html*) and Foundation Level Agile Tester (*https://www.istqb.org/downloads/send/5-foundation-level-agile-tester/41-agile-tester-extension-syllabus.html*). This was done to keep the definitions taught adhering to practices used in software test certifications. During the classes, participation will be stimulated so that everyone involved can raise questions, raise doubts and bring experiences they have had in the job market or research projects.

Bearing in mind that for a greater solidification of the proposed content, a practical exploratory testing activity will be carried out, using the *Selenium* tool (*https://www.seleniumhq.org*) and the Dojo Kata methodology, where one participant presents all the steps that must be followed to solve the challenge, while the other participants ask questions and suggestions [29]. The porfessor will be the guide for the task, however if any of the participants is available he / she can take over as a pilot, he / she should be rewarded in gamification when this occurs. Participants can suggest new test cases, suggest requirements, point out defects or raise improvements to the tested interface. The professor will take paper and pen so that the participants can make notes and later discuss the result. The teaching plan structure for this learning topic is shown in Table IV.

The gamification used in this learning topic will have the class types: classic lecture class and Kata-type Dojo. It will be supported by the game elements mentioned above. Thus, the guiding questions Q1 and Q3 will be answered.

TABLE IV. STRUCTURE OF TEACHING PLAN OF TOPIC 1.1

Introduction to software testing	
Competencies	**Practice and active methodology**
2, 4 e 5	Classic lecture classes Kata-type Dojo Gamification
Evaluation	
List of exercises on the syllabus. Individual and group evaluation of the Dojo Kata	
Tools	**Reference Material**
Selenium Paper Pen	Foundation Level Level Agile Tester ByeBug Project (http://www.inf.ufsc.br/~fabiane.benitti/byebug/)

B. *Software Development Concepts*

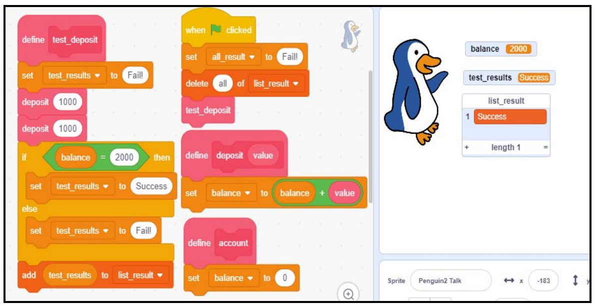
This teaching topic aims to provide common error identification concepts, good coding practices, code review and refactoring principles, aiming at keeping the code readable, skills that bring participants into ways that the development in an agile way is practiced in the industry.

Few classic lecture classes will be used, to sharpen the participants' learning. The teaching approach will have a more humanistic scope, with the participants at the center of knowledge. The teaching strategy chosen for this topic was a set of practical development activities using TDD, as this is an evolutionary approach to development, and a test must be written before writing enough code development to perform the test and carry out test and its refactoring [29].

Instead of individual practices, the teaching methodology selects again to use Dojos of programming, but this time of the Randori-type [19] and instead of selecting some conventional programming language, it was preferred to choose Scratch in view of the adherence of the tool with design patterns as analyzed in [30].

The Dojo technique is very efficient for group learning and to assist teaching software testing, allowing for a playful and competitive learning, which participants learn from each other [29]. So, was decided to create a challenges list related to the code development and testing of a banking service.

Some features that must be carried out by the code are: making a deposit, checking balance, making withdrawals, adding investments and checking the existence of a balance to make withdrawals. It is important to note that the activity will be carried out following the TDD flow using baby steps as can be seen in Fig. 2.



```
define test_deposit
set test_results to Fail!
deposit 1000
deposit 1000
if balance = 2000 then
    set test_results to Success
else
    set test_results to Fail!
add test_results to list_result

when clicked
set all_result to Fail!
delete all of list_result
test_deposit

define deposit value
set balance to balance + value

define account
set balance to 0
```

Fig. 2. TDD flow with Scratch for deposit functionality.

A greater possibility of realization of development practices exists using the Scratch tool, so during this teaching topic activities in addition to the Dojos Randori, a challenges list will be sent to the participants. Thus, the structure of the teaching plan for this learning topic can be seen in Table V.

TABLE V. STRUCTURE OF TEACHING PLAN OF TOPIC 1.2

Software Development Concepts	
Competencies	**Practice and active methodology**
2, 3 e 6	Classic lecture classes Randori Dojo Scratch challenge list Gamification
Evaluation	
List of development activities Dojo Randori group assessment	
Tools	**Reference Material**
Scratch	General Scratch Guide (https://resources.scratch.mit.edu/www/guides/en/EducatorGuidesAll.pdf)

The gamification in this learning topic will be the types of classes: Classical Lecture Classes, Randori Dojo and will be supported by the game elements mentioned above and the Q1, Q2, Q3 and Q6 guiding questions will be answered.

C. *Software Quality Strategies in line with Testing*

This teaching topic was developed in order to provide concepts for building testable projects, building agile evolutionary requirements and preparing agile test cases. The professor will act by asking pertinent questions, in order to guide the participant's learning, and solving possible doubts that occur during the class.

As in the previous topic, the number of classic lecture classes will be reduced to the level of concepts and to remember what was learned in the first topic. Thus, the focus of the teaching approach will also be centered on the participants, who will be divided in pairs, will receive developed software and will need to do the system and interface tests in search of bugs, later they will create and propose and build new requirements for the code, using the BDD (Behavior Driven Development) technique.

The programming language for the code that will be passed on to the participants will be Java, this choice was made by the large amount of plugins, libraries and tools used in agile development. This activity is the starting point of the project that will be carried out until the end of the teaching plan aiming of the participants having experiences that simulate the labor market and being able to develop a widely testable product, at that moment, the participants will be taught using the Eclipse tool with Junit and Cucumber plugins, and the Apache Maven tool.

The workflow for this teaching topic will be a sequenced activity list where the participants will perform the following tasks: exploratory tests in a cash flow system developed in Java language, creation of testing cases for correction, upgrading and new functionality requirements using Cucumber and Junit plugins, and tool management and build automation will be done with Maven tool. Thus, the teaching plan structure for this learning topic can be seen in Table VI.

TABLE VI. STRUCTURE OF TEACHING PLAN OF TOPIC 1.3

Software Quality Strategies in line with Testing	
Competencies	**Practice and active methodology**
1, 3, 4 e 6	Classic lecture classes Activities in pairs Gamification
Evaluation	
List of activities and evolution of the project	
Tools	**Reference Material**
Eclipse, Cucumber Java Maven Junit	Maven Guide (https://maven.apache.org/guides/getting-started/) Cucumber Guide (https://cucumber.io/docs/guides/)

The gamification in this learning topic will be the types of classes: classic lecture classes and continued challenges list that must be fulfilled during the classes. The guiding questions that will be answered are Q4, Q5 and Q7.

D. *Test-Oriented Development Project*

The last teaching topic covered will be the largest of this teaching plan proposal, which is the continuation of the activity that started with the requirements survey in the previous topic. In this topic there will be an immersive experience of software development oriented to tests, agile development methods, verification and validation of testable work products and coverage of software testability. The professor will have the role of a guide and will assist project participants with mentoring of the difficulties that will occur, it will be encouraged that at each class at least one pair present the progress of their task so that the rest of the class can collaborate with questions and opinions.

This topic will be focused only on practical software development activities based on the requirements obtained on last topic, an agile test plan will be built with the TestLink tool for storage and control of work products. Then, usability tests and non-functional tests will be carried out, in addition to estimating the time and risk of software development in the work products generated up to that moment.

Subsequently, the software will enter the stage of development, evolution, and integration. In some moments, the participants will have to deal with possible changes in requirements and new features. In this topic, participants must complete the software and deliver completed requirements accompanied by code coverage.

The workflow for this teaching topic will be: construction of a basic test plan to monitor the requirements and evolution of the work done by the pairs, complement of the test plan with test work products on usability and non-functional requirements. In addition to an approach to estimate construction time and risk using Testlink, software development using Eclipse, Maven and Junit, and the coverage test must be done using the Eclemma tool. The teaching plan for this topic can be seen in Table VII.

The gamification in this learning topic will be of the types of classes: there are challenges and will have the highest proportional score in the teaching plan. The guiding questions answered in that topic are Q5, Q6, Q7 and Q8.

TABLE VII. STRUCTURE OF TEACHING PLAN OF TOPIC 1.4

Test-Oriented Development Project	
Competencies	**Practice and active methodology**
1, 2, 3, 4, 5 e 6	Activities in Pairs Gamificação
Evaluation	
List of activities and evolution of the project	
Tools	**Reference Material**
Eclipse Cucumber Java Maven Junit Testlink Eclemma	Testlink Guide (http://testlink.sourceforge.net/docs/documents/end-users/manual.html) Eclemma Guide (https://www.eclemma.org/userdoc/)

E. *Guidance for Application of the Model*

As a way to guide professors who propose to adhere to the use of this model, some guidelines for the application of the teaching model are defined below:

- The teaching methodology must be centered on the participant as the actor of learning,
- The professor must be a facilitator for the teaching and learning process,
- The work to be done outside the class must be employed in such a way that the participant learns to solve problems and is encouraged to learn and bring doubts to the classroom,
- The professor, in addition to presenting the applications of theoretical and practical content, should also be a mediator, stimulate communication, provoke the participants to bring external knowledge to the classroom,
- The professor muste stimulate the participants' communication and negotiation skills.

At the end of the subject and the teaching plan, there will be a participatory feedback class to collect information, criticisms and improvements on the content presented, teaching strategies and the teaching plan used.

Topics such as gamification, tools used, material quality, main practices learned, references used, among others are addressed. Then all this interaction between the participants is recorded, so that later the audio transcription and a qualitative analysis of what was collected happen. All participants will be informed about this recording.

VI. EVALUATION

The evaluation of the documentation present in the sofware testing teaching model will be carried out by means of a focus group, which is a research method that encourages participants to reflect on the many aspects of an analyzed theme and its main intention is to improve the understanding of how people feel or think about the highlighted topic. For this research, the criteria for the selection of participating experts were: having teaching and professional experience in the Software Engineering area; holding a doctoral degree; having research publications related to the teaching of some Computing area.

At first, there was an analysis and individual evaluation of the model, through the electronic questionnaire that had the results that can be seen in Table VII.

TABLE VIII. EXPERT PANEL QUESTIONNAIRE

Questions	Answers		
Is the text and construction of the documentation adequate to the theme addressed by the model (Teaching of software testing)?	Yes	Yes	Yes
Is the text of the documentation clear and objective?	Yes	No	No
Does the text of the documentation present the minimum components necessary to specify a teaching model for software testing?	Yes	Yes	Yes
Does the documentation in its structure and style allow the change to suitability in another topic of computer education, in an easy, complete and consistent way?	No	No	No
Is the level of detail sufficient to provide an adequate basis for using the teaching model?	No	Yes	Yes

The experts considered the model to be suitable for teaching software testing, as the model had an instance of use and an academic syllabus in the same document, most experts did not find it clear what the purpose of the proposal was. Most considered that the model presented has what is a teaching model with enough detail to provide a learning base for software testing. Finally, the experts considered the model without a margin of adequacy for other computing topics other than software testing.

In a second moment of evaluation, there was a consensus meeting between the parties via videoconference where the experts made suggestions to enrich the model, such as: there is no library of assets to be used in the adoption of the model, the model is easy to use only for the subject presented, the model seems extremely prescriptive and its adaptation to different teaching environments does not seem trivial and moreover, some effort is needed to prepare the suggested projects and materials. This instance must propose teaching strategies. Several of these suggestions were considered when creating the next version of the model.

VII. CONCLUSION

The goal of this research is to contribute to the development of a model for software testing teaching using an agile context, active teaching methodologies and procedures used in labor market.

The software testing teaching model presented a set of approaches, techniques, methods, tools and resources that can be used for a teaching-learning process of software testing in an Agile Context, by a teaching plan developing an academic subject. Thus, this model measures to be a standard that can be used to guide the conduct of testing courses or subjects in the Computing area, to select instructional materials and to guide the actions of professors and obtained a favorable opinion from a experts panel for its use.

Regarding the research questions raised, about RQ I the answer is positive for the possibility of building a syllabus using active methodologies, practical tasks and agile practices used in labor market, it can structure a teaching plan that, governed by a syllabus, was evaluated by experts as suitable for teaching software testing. As for RQ II, the answer is partially positive, as the experts' panel evaluated that the model construction has the minimum necessary for teaching software tests; however, only after case studies in the classroom, the authors will be able to measure whether students have acquired the desired academic skills.

Software testing teaching and training model using agile context can be adhered to software testing teaching syllabus based on academic skills. The result was also positive, because the model was considered adequate to testing teaching in general regarding tests. These points can be seen as a mental exercise that articulates the use and application of a greater set of techniques and knowledge [9].

A. *Contributions*

The main contributions of this work are: the presentation of a teaching model geared towards tests with active teaching methodologies to be used correlated to playful teaching tools and practices to give an immersion of practical learning with a proposal aimed at development with tests.

We can also consider that the approach to create a teaching plan, linked to an existing subject to achieve academic skills is a proposal that can be used by other researchers in the computer education area.

B. *Limitations*

There are some limitations in this research, the model has been finalized, but there is still a lack of application of a case study in the classroom to measure its efficiency and effectiveness. Thus, only after that it will be possible to identify the functioning and gains that can occur in learning.

Another limitation would be that the model was considered highly prescriptive and difficult to adapt to other forms of computer education.

C. *Threats to Validity*

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 because the skills that served as a basis for building learning are the Ministry of Education in Brazil, thus the generalization of results is limited in case the skills to be used as a parameter are very different.

Another threat is that even if the teaching approach is aimed at practice and what is used in labor market, there is no guarantee that the skills and knowledge acquired will actually be reflected in a real environment of a software development team.

D. *Future Works*

The main future work is to carry out the case study of this model in the classroom, where this case study will be carried out in the second half of 2020.

It is also thought to make an adaptation in the teaching strategies for a case study to be fully used with the Scratch tool, however with participants who are in initial stages of testing and software development learning.

VIII. ACKNOWLEDGMENT

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

[2] B. S. Clegg, J. M. Rojas, and G. Fraser, "Teaching software testing concepts using a mutation testing game". In: 39th International Conference on Software Engineering. Buenos Aires, Argentina: [s.n.], p. 33–36. 2017.

[3] F. B. V. Benitti, "Evaluating Learning Objects for Teaching Software Testing". In: Nuevas Ideas en Informática Educativa. TISE, Chile. 2015.

[4] 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". Em 2015 International Symposium on Computers in Education (SIIE), 62–68. 2015.

[5] I. Elgrably and S. Oliveira, "Construction of a syllabus adhering to the teaching of software testing using agile practices". 50th Annual Frontiers in Education – FIE'20. Uppsala – Sweden. 2020., in press.

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

[7] 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.

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

[9] 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.

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

[11] A. Soska, J. Mottok, and C. Wolff, "An experimental card game for software testing: Development, design and evaluation of a physical card game to deepen the knowledge of students in academic software testing education". In:, 7th IEEE Global Engineering Abu Dhabi, United Arab Emirates. p. 576–584. 2016.

[12] P. Lauvas and A. Arcuri, "Recent trends in software testing education: A systematic literature review". In: The Norwegian Conference on Didactics in IT education. 2018.

[13] 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.

[14] H. T. Hung, "Flipping the classroom for english language learners to foster active learning". Computer Assisted Language Learning. v. 28, n.1, p. 81–96. 2015.

[15] P. H. D. Valle, E. F. Barbosa, and J. C. Maldonado, "A systematic mapping on teaching software testing". In: XXVI Simpósio Brasileiro de Informática na Educação. Maceió, Alagoas, Brasil: [s.n.], p. 71–80. 2015.

[16] C. M. C. de Oliveira, E. D. Canedo, H. Faria, L. H. V. Amaral, and R. Bonifácio, "Improving student's learning and cooperation skills using coding dojos (in the wild!)," in 2018 IEEE Frontiers in Education Conference (FIE). IEEE, pp. 1–8. 2018.

[17] G. Fraser, "Gamification of software testing". In: IEEE PRESS. Proceedings of the 12th International Workshop on Automation of software testing. [S.l.], p. 2–7. 2017.

[18] J. M. Rojas and G. Fraser, "Code defenders: A mutation testing game". In: IEEE. software testing, Verification and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on. [S.l.], p. 162–167. 2016.

[19] J. Rooksby, J. Hunt and X. Wang, "The Theory and Practice of Randori Coding Dojos". In: 15th International Conference on Agile Software Development, pp 251- 259. 2014.

[20] B. Estácio, R. Oliveira, S. Marczak, M. Kalinowski, A. Garcia, R. Prikladnicki, and C. Lucena, "Evaluating collaborative practices in acquiring programming skills: Findings of a controlled experiment". in 2015 29th Brazilian Symposium on Software Engineering, Sept pp. 150–159. 2015.

[21] S. J. Aguilar, C. Holman, and B. J. Fishman. "Game-inspired design: empirical evidence in support of gameful learning environments". Games and Culture, vol. 13, no. 1, pp. 44–70. 2018.

[22] N. Whitton, "Playful learning: tools, techniques, and tactics". Research in Learning Technology, vol. 26, 2018.

[23] L. N. Paschoal and S.R.S. Souza, "Planning and Application of Flipped Classroom for Teaching Software Testing". Renote, [s.l.]. v. 16, n. 2, p.606-615. 2018.

[24] A. C. Vasconcelos, G. L. A. Souza, S. A. B. Brainer, R. M. Soares, L. D. S. Barbosa, and P. I. S., "Teaching strategies through active methodologies". Brazilian Journal of Development, v. 5, n. 5, p. 3945–3952, 2019.

[25] I. S. Elgrably and S. R. B. Oliveira, "Gamification and Evaluation of the Use the Agile Tests in Software Quality Subjects: the Application of Experiments". 13th ENASE. Madeira,Portugal, 2018.

[26] I. S. Elgrably and S. R. B. Oliveira. "A Teaching Proposal or Testing Application with a Focus on Agile Methods Made Through Asset Mapping". 16th International Conference on Information Systemas & Technology Management. 2019.

[27] 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.

[28] Y. K. Chou, "Actionable Gamification - Beyond Points, Badges, and Leaderboards". Octalysis Media. 2015.

[29] J. Smith, J. Tessler, E. Kramer, and C. Lin, "Using peer review to teach software testing". In: IX annual international conference on International computing education research. Melbourne, Australia: ACM, p. 93–98. 2012.

[30] K. Amanullah and T. Bell, "Analysing students scratch programs and addressing issues using elementary patterns". in 2018 IEEE Frontiers n Education Conference (FIE), pp. 1–5. 2018.