Towards a gamified tool to improve unit test teaching

Matheus Marabesi*, Ismar Frango Silveira†
PPGEEC, Mackenzie Presbyterian University
São Paulo, Brazil
*Email: *matheusmarabesi@gmail.com, †ismarfrango@gmail.com*

***Abstract*—Due the constant evolution of software development and its growing complexity, more is being required from professionals to master different phases of the software development process, which includes the testing phase. This has an impact on the training of new professionals, since the undergraduate curricula is supposed to address this complexity. In this sense, software testing has its place in undergraduate CS-related courses, still the literature points out that the needed attention and importance is not given by the students and even by academic curricula. One of the possible causes that can be mentioned is the way of offering this content, which is usually part of a Software Engineering introductory course, perceived by students as something tedious and not important for their career. Thinking about this scenario, this paper describes the design and implementation of Testable.**

***Keywords*-unit testing; CS Education; gamification;**

I. Introduction

With the increasing evolution of software development complexity, in the professional or in the academic context, it is increasing the necessity to apply software verification and validation. Among the different kinds of tests in Software Engineering, there is the unit testing, which is dedicated to test individual units of code.

It is desirable that developers write unit tests. In general, these tests are performed per unit of code to verify its functionality. However this is not the reality by companies neither in the academic world [1] [2]. The curricular matrices of computer science have a reduced workload in software testing, under the premise of being enough for students to be prepared to perform properly the tasks related to software testing,which is not necessarily true.

This may lead students to think that software testing is a detail in the Software Engineering process. In addition, it can be hard for teachers to engage students for learning techniques to test software, since it requires a good level of abstraction and prerequisites. Besides that, common teaching strategies can have a negative impact on engagement, which motivates research involving game-based strategies, specifically gamification.

II. Gamification

Games are known by most people as a means of fun, a hobby or even a time to share with family and friends. Despite all the efforts of the academia to show that games can be used for other means than pure entertainment, there are still misconcepts and even some prejudice by the general public to advocate that games are limited to provide nothing more than fun, thus ignoring the potential of serious games, or other kinds of game-based learning, including gamification. There are games created specifically to teach and train professionals from various areas of knowledge, unlike games of entertainment, serious games are games developed with the aim of educating or training instead of pure fun[3][4].

Gamification has been applied in scale by several researchers and teachers in the academia as an alternative to the traditional method of teaching, in order to increase students engagement [5][6][7].

In general the term gamification has been gaining popularity by researchers in different areas [8], [9] and [10] define gamification as the use of game elements in a context other than a game. Such as the use of a card game in an educational context to teach the computer history [11]. In this context the cards are the element of the game and the history of computing the subject to be taught.

[12] have a broader definition, define gamification as the use of game mechanics, aesthetics, and game thinking to engage people, motivate action, promote education, and solve problems. According to the same authors it is necessary to differentiate what is a game, a gamification and a simulation to avoid any confusion between the definitions. Following with this reasoning, the characteristics of a game mentioned by the authors are illustrated in the following list:

- There is a defined gaming space where players agree to engage in gaming activities
- There is a clear start, middle and end to a game
- There is a defined winning state
- Players know when they or someone has completed the game
- A game usually has several game elements
- The games contain challenges, a mechanism for multiple attempts, some kind of rewards system, a clear goal that players work to achieve and a final goal

Gamification is commonly confused with games, though it has some key topics that differ, as seen in the following list. In gamification it is possible to use only one game element (although it is not restricted to), such as badges to engage a person to achieve a goal, whereas in a game more than one game is commonly used [12].

978-1-7281-4286-9/19/$31.00 ©2019 IEEE
DOI 10.1109/LACLO49268.2019.00013

- The intention is not to create an independent unit
- You can use only one element of the game to engage
- The intent is to use game elements to encourage them to engage with content and progress towards a goal

Finally, the simulation is the one that presents a greater degree of difference between game and gamification, whose aspects are:

- A Realistic Environment
- Players can practice behaviors and experience the impacts of their decisions

Simulation has the primary purpose of providing a realistic environment to facilitate learning. As an example, simulation is used by airlines to reduce cost and especially to train pilots with little experience, thereby reducing accident rate and procedural error when commanding an aircraft[1].To this end, the hardware and software suite must faithfully simulate what the pilot will face in the real world.

III. Game elements

In general, game elements are defined as a set of components in a game. Dynamic, Mechanic and Component are the main categories of game elements that exists in gamification [13].

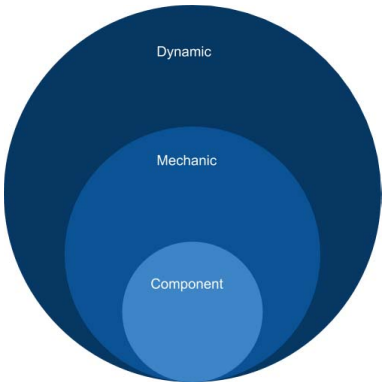

Figure 1. Game elements divided into categories.

The Figure 1 depicts a visual representation of each component and sections III-A, III-B and III-C follows a discussion on how they are related to each other.

A. Dynamics

Dynamic is a more comprehensive part of game elements, the one with the highest level of abstraction [13]. It is often performed by designers who have a more creative vision within the profiles that involve the development of a game.

It is through player's dynamic that the more general topics of the game are defined as the limitations of the player, the story line of the game, the progression of the player among others. However, the dynamics are not limited to these aspects, namely:

- Limitations (restriction or forced trade)
- Emotions (curiosity, competitiveness, frustration, happiness)
- Narrative (a consistent, continuous story line)
- Progression (player growth and development) Social relationships generating feelings of camaraderie, status, altruism

B. Mechanics

Software architecture is defined as decisions made in a software project that can not be reversed without a minimum level of effort, decisions that require an effort to change and are not easy to change in an afternoon [14].

Like software architecture, mechanics have elements that require an effort to be modified or changed once implemented. It does not mean that it would be impossible to change, only that these changes require a significant effort.

As a starting point [13] have defined a synthetic list with the mechanics that stand out, and according to the authors are the most important to consider for a gamification system.

- Challenges (puzzles or other tasks that require effort to solve)
- Chance (elements of randomness)
- Competition (one player or group wins, and the other loses)
- Cooperation (players must work together to achieve a common goal)
- Feedback (information on how the player is evolving)
- Acquisition of resources (obtaining useful or collectible items)
- Rewards (benefits for some action or achievement)
- Transactions (negotiation between players, directly or through intermediaries)
- Turns (sequential participation by alternate players)
- Win status (goals that make a winning player or group - State of draw or loss are related concepts)

C. Components

Finally, components are the last part of the whole that are game elements. In the same way that dynamics is the most abstract model of elements, the components are the opposite. They are the most concrete compared to dynamic and mechanic elements. User interaction is done through the components, to describe what can be accomplished within a gamified system or a game. The following list illustrates the fifteen most important components that should be present in a gamification. Like mechanics, it is not necessary to have all the components illustrated, it depends on the goal to be achieved [13].

- Achievements (defined goals)
- Avatars (visual representation of player character)
- Badges (visual representation of achievements)

[1]http://www2.fab.mil.br/musal/index.php/projeto-av-hist/62-projeto-av-hist/470-os-primordios-dos-simuladores-de-voo

- Boss Fights (especially difficult challenges at the culmination of a level)
- Collections (sets of items or badges to accumulate)
- Combat (a defined battle, usually of short duration)
- Content unlocking (aspects available only when players reach goals)
- Gift (opportunities to share resources with others)
- Leaderboards (visual displays of player progression and achievement)
- Levels (steps defined in player progression)
- Points (numerical representations of the progression of the game)
- Missions (pre-defined challenges with goals and rewards)
- Social graphs (representation of the social network of the players within the game)
- Teams (defined groups of players working together for a common purpose)
- Virtual Goods (game assets with perceived value or real money)

IV. Software testing in the context of CS Education

The task of software testing is the part where teams must devote themselves exclusively to this task, using appropriate software defect control tools and ensuring the correct operation of the requirement. Given it importance for many software processes, it is expected that such a subject to be part of CS curricula when dealing with undergraduate education.

Among many types of tests, Unit test deserves a significat mention, given the easiness of its integration with first-years courses in CS curricula, including courses related to basic programming, or even most advanced Software Engineering courses. Because of its flexibility, this kind of test is being targeted in this paper.

A. Software testing

The task of software testing is the part where development teams must devote time exclusively to this task, using appropriate software defect control tools and ensuring the correct operation of the requirement. However, this is not what we find in the literature, although the subject is approached by several authors [15] [16].

The definition of software testing according to [16] is the task of showing that a software does what it intends to do, and in addition, discover its defects before putting the software into use. According to the same author, we can divide the objective of testing software into two parts:

- 1) Demonstrate to the client and developer that the software has the specified requirements.
- 2) Expose unwanted behavior.

Through these two definitions we have an overview of what we find in the software testing process. The first topic is to demonstrate to the client that what is being built is exactly what was previously specified. The client in this sense dos not necessarily need to be a person, because it often becomes an entity, such as a company.

The second topic has an opposite objective, exposing an undesirable behavior and even possible failures given a set of data, behavior that should also be explicit in the software requirements, even if it is undesirable.

B. Unit test and Test driven development (TDD)

Unit testing is the type of test that is created and executed by the programmer at the time the software code is written, this type of test is the opposite of the black box and white box test that is often performed by a professional dedicated to this task, and does not write the code in question. [16] defines unit testing as the process of testing different components of a program, such as methods and classes.

Unit Test was disseminated through the TDD methodology where the programmer begins to write the test code before the actual software code [17]. Second [18] TDD is an awareness of the gap between decision and feedback during programming.

[18] mention the gap between feedback and the issue in having a quick response at the time that the code is written. Often the writing process by the programmer follows three basic steps: The first is writing the code, the second is to test and the third the refactoring in case something is wrong or there is a need to change the code.

TDD can be considered a response to the evolution of the graphical interfaces and the evolution of the flows to perform tasks in complex systems, the programmer finds himself ahead of time performing extra steps to get to the point where his code can be tested, as a result the feedback is late, just to get a sense if the code is working as expected, which directly impact the late delivery of the software changes.

The TDD mantra depicted in the Figure 2 defined by Kent Beck follows three distinct phases:

- Red - Write the test code first, thinking about how it should behave.
- Green - Make the test pass with the least possible change.
- Blue - Refactor, remove any duplicate code and make small improvements.

The first phase of the TDD (red) is the first step to be considered when programming with unit test. For some developers it is strange to write the test code before the ”production” code. Production code is a term used for code that will effectively build the desired algorithm to perform a given task. The barrier here is the paradigm shift, which may be more complex for those with more programming experience. Years of accumulated programming experience makes the shift to a new paradigm costly.

On the other hand years of experience can help the programmer to make the shift, with experience comes the

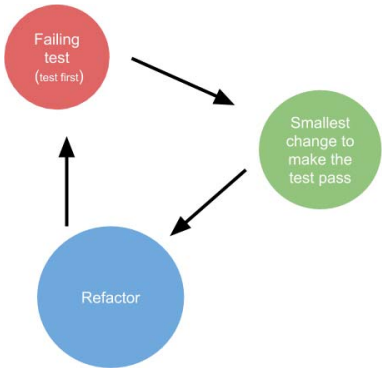

Figure 2. TDD flow. Based on [18]

mastery of a programming language and as a result the task of writing tests becomes less complex because the focus will be writing the test code, without having to do external queries to solve a doubt of syntax.

Then enters the second stage of the cycle, writing the production code that makes the test pass with the least possible change. It is important to note: "the least possible change ", this step focuses exclusively on having the test written to pass. This step is what [18] defines as baby step.

Baby step is the act of making a small change that reaches the goal as well as the steps of a baby, which are short still makes the baby to walk. Particularly the stage of the bay step is one of the most valuable technique using TDD, changing the smallest part of the code required for the test to pass provide programmers a moment of self reflection while programming.

Experimenting with this step can also have an undesirable effect on the cycle, since more experienced programmers tend to use higher abstraction, without the need for it. This behavior is known as over engineering [19].

Finally the third and final part of the cycle is refactoring. Refactor is the term used to describe the activity of changing a system's existing code without changing its external behavior, and making improvements to avoid errors [20]. Although it is a technique often used by programmers, refactoring existing code in systems that do not have automated tests become a costly and risky task. As it is impracticable to guarantee that the refactoring does not produce side effects in different parts of the system.

At this point in the cycle we should change, if necessary, the structure of the production code to fit the goal to be achieved. Often this is the step that fix elements introduced into the code by the previous step, for instance changing hard code values to variables.

C. Software test teaching

The software test teaching in undergraduate courses in the computing area has a shorter workload compared to other topics. This may reflect the reality of the industry as well as in the academia, since in practice development teams do not engage as expected in the testing phase of software [1] [21].

[1] presents the national reference curricula, among them CEEInf 1999, SBC 2003, SBC 2005 and international IEEE 2004, ACM, AIS and IEEE 2006. Which of those in the national curricula presented here, all mention the subject Software testing as part of the discipline of Software Engineering or as part of other disciplines. Still according to the authors, there is no detail on each item that should be addressed in this subject. International curricula, such as the ACM, makes a suggestion about how the software testing subject should be. The ACM suggests that software testing should be an exclusive subject, yet it leaves the details to be addressed by the university.

Although the subject of software testing is a subject addressed by several authors, recent work points to the lack of dedication to this stage of software development in the industry. Especially TDD [22], which is a practice adopted by programmers of various programming languages. Still TDD may not be a day-to-day practice for developers. [23] monitored 2,443 software engineers for 2.5 years in their integrated development environment (IDE) and pointed out that half of that amount does not perform test on their application as they develop, it shows the opposite desired behavior from TDD, which requires a test suite running after each code change.

In this sense, there are two software testing categories: the one executed by professionals dedicated to this task (IV-A), and the programmer-oriented test while developing the software (IV-B). The literature points out that no matter which of the approach is used to test software, both compared to other topics of computing have not enough attention dedicated to the testing phase.

V. Testable conception

Testable is a gamified tool designed (and not limited to) to be used in face-to-face education in undergraduate CS-related courses. The following list presents the topics that are covered in the Testable tool.

1) **Introduction: Unit Test - Introduce the unit test base concepts for the user. (Such as functions - the smallest piece of code)**
   a) Unit test applied to structured programming - Exemplification of the use of unit test with the paradigm, with examples of functions that are testable.
   b) Unit test applied to object-oriented programming - Use the concepts presented in the previous session adding the use of classes.
2) **Introduction: TDD - Introduces the TDD technique to the user. Exemplifying what TDD is.**

a) Red - First stage of the TDD cycle, to exemplify the purpose of writing the code prior to the production code.
b) Green - Second stage of the TDD cycle, just change enough code, so the unit test passes. Changing state from red to green.
c) Blue - Third and final stage of the TDD cycle, exemplify the purpose of refactoring (if necessary) and the confidence to change the code and ensure that there are no side effects.

A. *Testable history and Ludic world*

The playful universe of testable consists of the main character, a charismatic and intelligent insect, called Buggy (Figure 3), who dreams of being recognized as a great programmer, yet he is not well seen within this world, since it is ironically a bug.

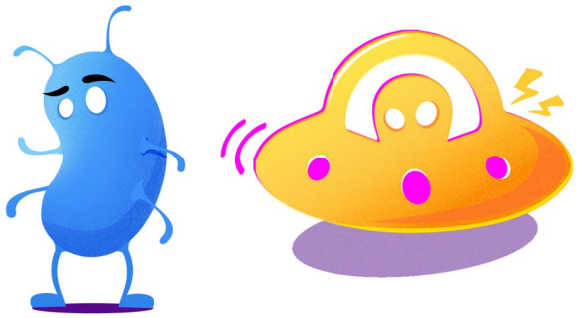

Figure 3. Main character buggy on the left and secondary main character on the right.

Today he has only a few months that write programs, and his favorite programming language is javascript. Although some more experienced programmers have some kind of antipathy with Javascript, little Buggy does not care.

Buggy is very bothered by the overvaluation of aliens in the programming market, as they are always stereotyped as the smartest in the world, as many have a partnership with Nazah[2], and are always launching trends in super-programming for the space market.

Now Buggy needs to show that he can be a great professional. He is developing a project, and as he is implementing new functionalities errors appear in other parts of the system that he didn't touch. After many hours spent trying to debug the problem, Buggy discovers that there are unit tests, and although they are important, they are barely used.

B. *Testable mechanics*

This section of the paper presents the division of the Testable content tool into gamified elements in order to build a progression for the user and which elements to use.

[2]Nazah makes a analogy with the real NASA - The National Aeronautics and Space Administration

The task of building a rich experience for the user is arduous and requires different parts of the gamification process to be done [24].

To ease this process [25] created a set of thirty-five cards[3] to help creating the gamified experience in a simple and effective way. His toolkit for developing a game based experience is named "toolkit for Gamification Design" and is based on cards and divided into colors, as presented below:

- Pink - Onboarding
- Yellow - Late Onboarding
- Orange - Midgame
- Blue - Late Midgame
- Green - Endgame
- Purple/Epic - Everlasting experience

In the version two of the toolkit, for each item in the list presented, there is a set of cards that are distributed as follows: the cards in pink, yellow, orange, blue and green have six cards each[4].

The first version of the toolkit developed by [26] is directly linked to the version two, since the base and theoretical foundations are extracted from the first version and used extensively in version two. Version one differentiates itself by not having a set of cards to be used in the definition of the gamified experience, instead of cards, four tables are used [26].

[25] suggests in total three steps to use the toolkit, the first one is to get the cards and print them. Then the suggestion is to divide the cards by color, since the colors used represent a step in the process of designing he gamified experience. The last suggested step is to make different combinations to create new experiences. The experience for the Testable tool is shown in Figure 4.

The resulting experience is as follow: A tutorial to introduce the Testable experience along it's learning process, experience points, an in-game currency, random rewards based on the user progress, rewards achieved by sharing the tool via social media and epic challenges for students that want to extend the experience.

The cards chosen form a linear combination in the evolution of the user in the gamified experience. The cards itself does not represents how the implementation should work, instead the cards represent the elements that the experience should have. In this sense Testable uses the elements Experience Points, In-Game Currency, Random Rewards and Free Lunch follows the user during the whole gamified experience, Tutorial and Epic Challenge cards do not. Besides that Testable focus on an individual experience, each user should have its own account, as it is needed to track (section V-C) all the items pointed by the cards.

[3]http://www.epicwinblog.net/2013/10/the-35-gamification-mechanics-toolkit.html

[4]Cards of the "toolkit for Gamification Design" ordered by color available at https://drive.google.com/file/d/1EwsBALfmyGK0QWuiGjogqIdrUWEhb5Qn/view

Figure 4. Cards combination based on [25] strategy

The gamified experience so far brings a broader approach, such as which game elements to use and how the progress should be. Therefore the specific experience on unit test is not mentioned, which makes sense because the toolkit is meant to be used by different subjects.

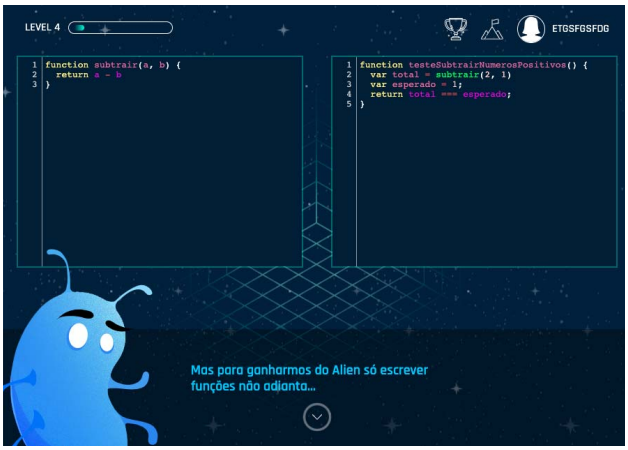

Figure 5. Testable main scene, on the left box the student fill in the source code, on the right side is the test code.

Furthermore the challenge in how to approach the unit test subject remains. The Figure 5 depicts the basic approach suggested, on the left the user types the source code and on the right the test code. Once this is defined the problem is how to make sense of the code that the user is typing and also track if the source code and the test code meets the requirements - requirements that are defined for each source code and test code. To overcome this, the strategy used is a parse followed by predefined testes cases.

Code platforms interview[5] and applications that aims to improve users programming skills[6] applies the same approach, the user writes the source code for a given problem, once the user is satisfied with the code, the platform has a predefined set of tests that acts against the code to verify its functionality. Testable uses the same approach, for each user interaction on both, left or right side (where the source code and test code is written as depicted in the Figure 5), the flow depicted in the Figure 6 is executed.

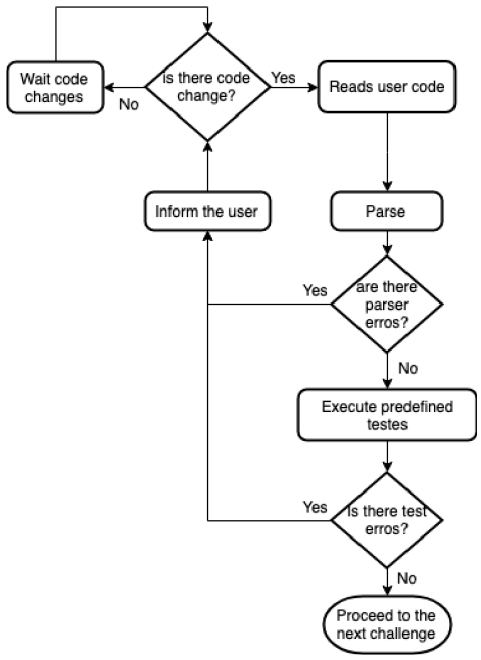

Figure 6. Execution flow for each user level.

The feedback for errors is given right below the boxes (Figure 5), to help users to identify the root cause more quickly, so if the error is in the source code, the error appears below the left box, and if the error is in the test code it shows below the right box.

Testable is intended to run on any modern web browser [7], the subset of technologies applied to develop the gamified tool are based on the web technologies: HTML, CSS and Javascript. Though, to achieve the final result, which is the code that the browser can run, different tools are needed, such as ReactJS[8] (a library to build web interfaces), nodeJs[9] and webpack[10]. The persistent layer is provided by Firebase, a real time database [11] that avoids the need to build a server

[5]https://www.hackerrank.com
[6]https://www.codewars.com
[7]Modern browsers target in this paper are considered to be with score 69 or higher based on https://caniuse.com browser score
[8]https://reactjs.org
[9]https://nodejs.org
[10]https://webpack.js.org
[11]https://firebase.google.com

side service.

C. Testable tracking system

In order to evaluate the Testable tool - to understand user behavior [28] and if it is effective or not, a tracking system is need to collect data. For that to happen the tool tracks four main events fired by the interface: on click, when any click on the interface is made by the user, on change, when any code is changed and on loaded or finished any section. The click event to be captured and logged as a trackable interaction must be one of the following: The user level component, the user menu (where the user can access the options menu and logout) and finally the user guide (depicted by Figure 5 as an arrow down). The change event is captured whenever the user changes the code, be it source code or test code. Finally the on load and on finished are used automatically by the gamified tool, whenever a user starts or finishes a section an event is tracked. Every event is tracked with a timestamp mark, and constrained by section and user as the Table I presents.

Table I
TRACKING INFORMATION COLLECTED FOR EACH USER ACTION

Tracked item	Description
Action	Describes the action being tracked, on load, on finished, on click, or on change.
Section	The section from where the tracking system is tracking the event, acceptable values are related to the content described in the section V.
Timestamp	The moment when the tracked event happened. To prevent timezone issues the timestamp format is used.

Such tracking system allows to understand how the user is behaving while using the tool. The collected data and interpretation is going to be crucial to evaluate if the tool is effective or not. Hopefully the tracking system is going to answer questions as: How long did the user stay in the tutorial section? Unit test section?, Is the user going straight to the next section or is he exploring the interface?.

In order to track broader informations that is not directly related to the gamified tool, Google Analytics [12] is used. Through Google Analytics information such as which browser is the user using, which operating system and even the network provider is possible to be tracked, besides that an extensive list is described by [29].

[12]https://analytics.google.com

VI. FINAL CONSIDERATIONS AND FURTHER WORK

The paper explored the universe of serious game and gamification, concepts which are often misinterpreted as being the same. However, the two approaches have divergences and are used for different purposes. The two approaches are used within and outside the academic world according to the literature presented. On the one hand there are serious games, games created with focus beyond pure entertainment, on the other hand there is gamification that uses elements of games in a context that is not a game.

Towards this definition this paper described a gamified tool to improve unit test teaching in undergraduate education, aimed to improve the student engagement. The tool is under development [13], focused on the gamification elements that can engage the student experience. Once the development is done, the goal is to apply the tool in an ongoing course that has software testing subject and evaluate if the tools has any impact, be it positive or negative.

REFERENCES

[1] Benitti V. B. F., Albano L. E.: Teste de Software: o que e como ensinar?. In XXXII Congresso da Sociedade Brasileira de Computao (CSBC) Itaja, Sata Catarina, p. 1-10 (2012)

[2] Beller M., Gousios G., Panichella A., Proksch S., Amann S., Zaidman A..: Developer Testing in the IDE: Patterns, Beliefs, and Behavior. In IEEE Transactions on Software Engineering, pp. 261 - 284 (2019)

[3] Michael R. David, Chen Sande.: Serious Games: Games that Educate, Train and Inform. Thomson Course Technology PTR, Boston, MA (2006)

[4] Adams E., Dormans J.: Game Mechanics: Advanced Game Design. New Riders, Berkeley, CA (2012)

[5] Lima T., Campos B., Santos R., Werner C.: UbiRE: A game for teaching requirements in the context of ubiquitous systems. 1. ed. In XXXVIII Conferencia Latinoamericana En Informatica (CLEI) (2012)

[6] Gmez-lvarez C. M., Snchez-Dams R., Barn-Salazar A.: Trouble hunters: A game for introductory subjects to computer engineering. 1. ed. In XLII Latin American Computing Conference (CLEI) (2016)

[7] Marques R. A. F., Miranda de C. L., Menezes de C. B., de Cunha E. E..: Karuta Kanji: Jogo educacional para estudar e praticar vocabulrio com Kanjis da lingua japonesa. 1. ed. In Latin American Computing Conference (2015)

[8] Matallaoui A., Hanner N., Zarnekow R.: Introduction to Gamification: Foundation and Underlying Theories. In Progress in IS. Springer, pp 3-18 (2016)

[9] Deterding S., Khaled R., Nacke E. Lennart, Dixon Dan.:Gamification: Toward a definition. In ACM CHI Conference on Human Factors in Computing Systems, pp. 12-15 (2011)

[13]The development is open source and the code is available on Github: https://github.com/marabesi/testable

[10] Brockmann T., Zarnekow R., Robra-Bissantz S., Lattemann C., Stieglitz S.: Gamification: Using Game Elements in Serious Contexts. In Progress in IS Springer, pp. 19 - 29 (2017)

[11] Santos O. C. J., Figueiredo S. da K.: Computasseia: Um Jogo para o Ensino de Histria da Computao. In XXXVI Congresso da Sociedade Brasileira de Computao pp. 2026 - 2035. Porto Alegre (2017)

[12] Kapp M. K.: The Gamification of Learning and Instruction, Fieldbook - Ideas into Practice. Wiley (2012)

[13] Werbach K., Dan H.: For The Win - How Game Thinking Can Revolutionize Your Business. Wharton Digital Press, Philadelphia, PA (2012)

[14] Brown S.: Software Architecture for Developers. Leanpub (2014)

[15] Pressman R., Maxim B.: Software Engineering - A Practitioner's Approach, 8. ed. Mc Graw Hill Education, New York, NY (2014)

[16] Sommerville I.: Software Engineering Tenth Edition. Pearson Education Limited, England (2017)

[17] JANZEN, D. Test-Driven Development: Concepts, Taxonomy, and Future Direction. 1.d. [S.l.]: IEEE Computer Society, (2005).

[18] Kent Beck, Bookman, TDD - Desenvolvimento Guiado Por Testes (2010).

[19] Kerievsky Joshua.: Refactoring to Patterns. Addison Wesley Signature, pp 43 (2005)

[20] Martin Fowler and Kent Beck and John Brant and William Opdyke and Don Roberts: Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional, 1. ed. Addison-Wesley Professional (1999)

[21] Sousa Virgnia Bezerra, C. I. M., Coutinho, Emanuel, Santos Ismayle De Sousa.:iTest learning: Um jogo para o ensino do planejamento de testes de software. In XX Workshop sobre Educao em Computao (WEI), (2012).

[22] Kent Beck.: Test Driven Development: By Example. In Addison-Wesley Professional, (2002).

[23] Moritz Beller, Georgios Gousios, Annibale Panichella, Sebastian Proksch, Sven Amann, Andy Zaidman.:Developer Testing in the IDE: Patterns, Beliefs, and Behavior. In IEEE Transactions on Software Engineering, (2019).

[24] Victor Manrique: Gamification Design Framework: The SMA Model (2013)

[25] Manrique V.: A simple and easy to use toolkit for Gamication Design by @victormanriquey. In Epic Win Blog (2013)

[26] Victor Manrique: The 35 Gamification Mechanics toolkit. In Epic Win Blog (2013)

[27] Banks A., Porcello E.: Learning React: Functional Web Development with React and Redux, O'Reilly Media (2017)

[28] Atterer Richard, Wnuk Monika, Schmidt Albrecht.: Knowing the User's Every Move: User Activity Tracking for Website Usability Evaluation and Implicit Interaction. In Proceedings of the 15th International Conference on World Wide Web, (2006)

[29] Clifton B.: Advanced Web Metrics with Google Analytics. 3. ed. Sybex (2012)