Towards Model-Driven Gamified Software Modelling Learning

Alfa Yohannis*, Dimitris Kolovos, Fiona Polack
Department of Computer Science
University of York
York, United Kingdom
Email: *ary506@york.ac.uk

***Abstract*— Motivated by the success of gameful approaches in different fields, this research harnesses the engaging nature of games combined with the effectiveness of pedagogy and the automation of Model-driven Engineering to propose a framework for model-driven gamified software modelling learning. It is a framework for tutors to create software modelling learning patterns, which later can be transformed to generate software modelling learning games for learners to play. This paper presents the motivation behind the initiation of the framework as well as the problem analysis and the solution overview in realising the framework. The framework is then demonstrated to show how it works. Two forms of assessments are then presented as the evaluation of the framework.**

***Keywords*—model-driven, gameful, software modelling, learning, framework**

I. Introduction

Gameful approach, whether gamification [1], serious games [2], and digital game-based learning [3], have been proven to have significant positive impact for a variety of purposes, such as learning, skill acquisitions, engagement, psychological supports, socialisation, etc. [4], [5]. Duolingo, an application for learning new languages (https://www.duolingo.com/), and Re-mission, a game to learn to deal with cancer (http://www.re-mission.net, are two popular examples of the gameful approaches. On the other side, the application of gameful approach in software engineering has not been studied extensively [6]. This encourages us to apply it in software modelling learning, an area which has received little attention for the application of gameful approach so far.

Some works [7], [8], [9], [10] have been conducted to use the gameful approach in software modelling context, and each of the works addresses different problems using different approaches and game elements. This fact challenges us to develop a more generic approach in addressing software modelling learning so that tutors do not need to develop individual games from scratch only to deal with certain topics. Therefore, we propose a framework for teaching and learning software modelling–a framework for tutors to create software modelling learning patterns, which later can be transformed to generate software modelling learning games for learners to play with.

The paper is organised as follows. First, this paper is started with an introduction to the background and motivation of our work. In Section II, some problems that challenge us in realising the framework are identified and analysed. After that, in Section III, an overview of our solution–the framework–is described. In Section IV, the framework is demonstrated to show how it works from the creation of a modelling language to playing it in a game. Subsequently, in Section V, the plan to evaluate the framework is presented and discussed. The paper is then closed in Section VI with the presentation of related works to position our study in the context of gameful approach in software modelling learning.

II. Problem Analysis

In order to engage continuously in a learning activity, learners have to be introduced to new more difficult challenges once their competence grows [11]. In facing the challenges, learners need to be given supports and then reduced step by step until they are capable enough to solve the challenge by themselves as suggested by scaffolded learning [12], [13]. The scaffolding can be expressed through procedural instructions, clues, examples, small activities, and pre-made models. Therefore, in order to master software modelling, learners should be exposed progressively from simple to complex modelling scenarios but without neglecting the supports until they reach desired competence. These approaches should be able to be expressed by tutors in their learning activity design.

The main challenge addressed in this work is transforming software modelling learning activity into gameful activity. Gameful design steps [14] suggested that to apply gamification the core activity should be identified and then gameful modification could be implemented. Modelling is the process of creating an abstraction of reality or concept, and usually, the abstraction is displayed in the form of visual representation (e.g. symbols, graphs, diagrams, drawings). The activity 'creating' and its synonyms (e.g. constructing, building, developing) are activities that fall into the category of creating in Bloom's taxonomy [15]. The LM-GM framework [16] maps the Bloom's creating category into the design, editing, and planning game mechanics whose activities are also performed in modelling. Thus, design, editing, and planning activities in modelling are inherent can be used as game mechanics as well. However, since we scope this research only to modelling in 2D diagrams, constructing and editing model in 2D diagrams are selected as the game mechanics of the software modelling learning.

Therefore, as gameplay, learners are asked to construct or modify diagrams so that the diagrams become consistent with a given problem description and instruction. They do not have always to start modelling from empty, but could also from existing, incomplete diagrams that are premade for them to help them focus on core concepts and activities being taught[14]. We name this kind of play as construction gameplay. Some other forms of play are also feasible to be implemented. For example, learners are provided with a problem description and a number of diagrams reflecting candidate solutions and needs to select the correction solution(s). Another form is learners are given with a diagram and number of statements in natural languages and needs to choose the correct interpretation(s) or statement(s). We called these two forms of play as multiple-choice gameplay.

For the construction gameplay, a graphical editor through which the user can construct/modify solutions is needed. Ideally, the editor should be fully integrated into the game to provide live feedback and an immersive experience (i.e. an alternative would be to ask the user to create/edit models in an existing modelling tool and upload them to the game). If the graphical editors are wanted to be embedded into the game, a standardised way to define the abstract and graphical syntaxes of supported languages is required.

There are more than one ways to assess consistency and correctness for the construction gameplay. One option is to require that the provided diagram is a 1:1 match with a reference solution given by the game designer. A more relaxed approach is only to require that the provided solution satisfies a number of constraints. Another option is to require some sort of semantic equivalence between the two models (e.g. class diagrams that are consistent with the same object diagram). Moreover, tutors need to be able to define different types of levels and link them up to assemble complex games.

Rewards are a major element of learning games allowing users to develop a sense of achievement, show off their new skills, compare themselves against other users, etc. So far, there are two types of rewards in this game, positive reinforcement and achievements. Positive reinforcement is presented in the form of sounds, visual effects, or encouraging texts to give feedback to players whether they are making good or bad actions while playing. Positive reinforcement is intended to improve the self-efficacy of players by keeping them informed of their progressing whether they are heading in the right direction or not, so they can decide and continue to execute their next actions [17]. Another reward is a list of a learner's achievements. Basically, the list acts like a 'CV' for a learner. It informs the names, types, and numbers of levels and learning patterns that they have been completed. Since a learning pattern usually addresses a concept in software modelling, completing the pattern could be used as base to claim that the learners have acquired the competence–knowledge and skill–for that concept which is rewarding to them [17]. Building a learner's competence also means strengthening their intrinsic motivation [18] which could lead to meaningful gaming activity [19].

III. Solution Overview

A solution, a web-based framework has been developed to address the problems presented in the Problem Analysis section. The framework implements a metamodel-annotation-based approach for defining the concrete syntax of the modelling language. The metamodel is written in Emfatic (https://www.eclipse.org/emfatic) and Eugenia-like annotations [20]. In this way, tutors can write different types of annotated metamodels to produce different types of visual modelling languages.

The annotated metamodel then transformed using Epsilon [21] and EMF [22] to generate the Java codes of the metamodel as the backbone codes for further model operations. The transformation also produces Javascript codes to display the metamodel's visual elements in a palette of a web-based graphical editor built on top of the MxGraph framework (https://jgraph.github.io/mxgraph/). After the generation, tutors can create models using visual modelling language defined by the metamodel.

A modelling language is also created for tutors for defining and linking up different types of levels. We name the models constructed by this modelling language as learning pattern models (for example, see Fig. 2). Currently, elements that are supported by the modelling language are activity, objective, model, start, end, transition, and link. The activity element represents activity that is usually found in a learning process. In an activity, tutors need to define lesson, instruction, and the name of the metamodel of intended learning. They also need to define one or more objective elements that learners have to meet in an activity in order to complete it. A model element is an element that refers to an existing model that already created by tutors or a model that will be produced by an activity. The model is used as the base model when learners start an activity, so they don't have to create the model from the beginning. Start and end elements indicate where a learning pattern should start and ends. Transition element is a directed edge that connects an activity to another, and link element is also directed edge that connects a model to an activity and *vice versa*.

Learning pattern models are then consumed by a model-to-text transformation to generate a complete playable game. The framework can host a number of generated games and provides features such as authentication, progress management, learning pattern inventory, model inventory, etc. The generation also produces an EVL [23] skeleton for each level where tutors define constrains and operations for validation to determine whether the level has been completed, all its objectives have been met, or not. Currently, only the construction gameplay is supported to be played at each level, and the correctness of solutions is assessed using the EVL file. The remaining level types and validation approaches, as well as reward mechanisms, are under implementation.

Some possible extensions that are planned for the framework are as follows. First, a fine-grained logging framework that will allow game designers to replay solutions and understand how users interact with the game. Second, an automated model mutation to automatically generate incorrect alternatives in a-type levels. Third, a facility that monitors the progress of learners and adapts the game accordingly (e.g. displays fewer instructions, allows fast learners to skip levels, or recommends other related games to play).

IV. Demonstration

In this section, an example is presented to demonstrate the use of the framework. The example is to show the of teaching the concept of substates in UML statechart modelling. In this demonstration, designers are assumed to have proficient knowledge and skills of statechart diagram while learners are assumed already understand the concept of states, transitions, start state, and final state. The following subsections are the order of the statechart from its definition to be fully used in a game play.

A. Define Statechart Metamodel

For the framework to support modelling in statechart diagram, designers should define its metamodel first. For example, the definition of Start State derived from Node class is displayed in Listing 1. Other basic elements of statechart diagram, such as states, transitions, end states, etc., are also defined but not displayed in the figure.

Listing 1: A definition of Start State derived from Node class using Emfatic and Eugenia-like annotations.

```
abstract class Node extends Entity {
  ref Edge[\\]#source outgoing;
  ref Edge[\\]#target incoming;
}

@node(label="name", dhape="startState", whiteSpace="wrap",
     html="1", fillColor="#000000", width="30", height="30")
class Start extends Node {
}
```

B. Create Statechart Model

Using the MxGraph-based graphical editor (Fig. 1), designers can now create statechart models. The editor has a pallete on the left side that contains elements of statechart's elements which they can drag and drop into a drawing area on the centre where they can arrange the elements to construct statechart models. The editor also has a property panel on the right side to modify the attributes and appearance of the models. Models that have been created can be saved and loaded again for further operations. The editor also can be used to create models as input/base models to be embedded in learning activities.

C. Create Learning Pattern Model

To teach and learn a concept, designers should design a learning pattern that consists of ordered learning activities. In this example, a learning pattern to learn the concept of substates is created (Fig. 2). The case that is selected is an electrical fan that has two main states, Idle and Blowing. Learners will be asked to modify the Blowing state, so it becomes a composite state that has several substates that represent the different speed of blowing. The learning pattern

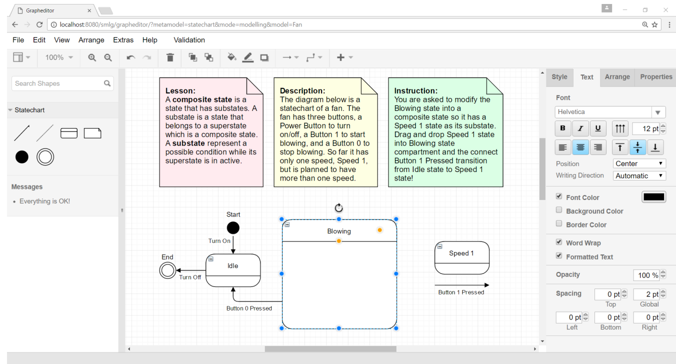


Fig. 1: The MxGraph-based graphical editor to create statechart models.

has three activities, namely One-Speed Fan, Two-Speed Fan, and Tree-Speed Fan. The activities are arranged *per se* to accommodate Flow state [11] so learners can start learning from the easiest activity to the hardest one. The activities are explained in more detail in the next following subsections.

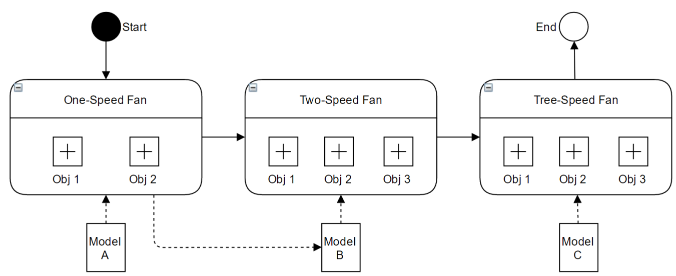


Fig. 2: A learning activity pattern for learning the concept of Substates in state-machine modelling.

Each activity has a lesson and instruction properties. Lesson contains an explanation about the concepts that are being taught, and instruction contains commands or questions that learners need to execute or answer. In the process of satisfying the instruction in each activity, one or more objectives should be met by learners to move to next activity. Also, each activity can consume existing models (Model A, B, and C in Fig. 2) as its base models, so learners do not have to create a model from scratch and produce a model (Model B in Fig. 2) to be used in its next activity. Each of the models also has a description property to describe itself which is very useful for learners to understand about the model. An example of the lesson, model description, and instruction properties of the One-Speed Fan activity (Fig. 2) is displayed in Fig. 3.

1) One-Speed Fan Activity: In this activity (Fig. 4), learners are introduced to one substate only. The activity starts with a base model, and learners are required to modify the base model (Fig. 4a) to meet the target model (Fig. 4b). The base model corresponds to Model A in Fig. 2, which refers to an existing base model and will be loaded once the activity is executed, so learners do not need to create the model from the start.

The example case in this activity is a fan that has three

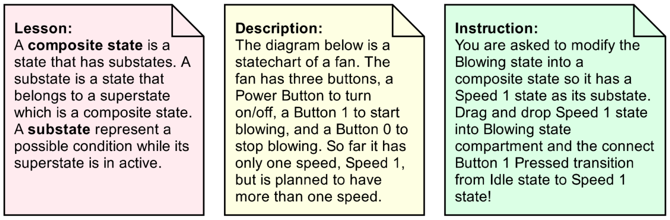
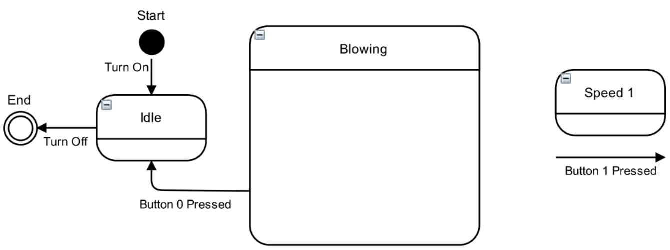
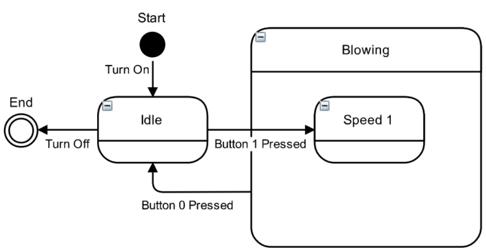


Fig. 3: Lesson, model description, and instruction.

buttons, a Power Button to turn on/off, a Button 1 to start blowing, and a Button 0 to stop blowing. So far it has only one speed, Speed 1, but is planned to have more than one speed. Learners are asked to modify the Blowing state in Fig. 4a into a composite state through moving the Speed 1 state into the Blowing state compartment as well as to connect the Button 1 Pressed transition from the Idle state to the Speed 1 state. Since in Fig. 2 this activity is designed to have only two objectives, the two objectives are adjusted and defined as follow: Objective 1 ”The Blowing state contains the Speed 1 substate” and Objective 2 ”Button 1 Pressed transition connects the Idle state to the Speed 1 substate”.



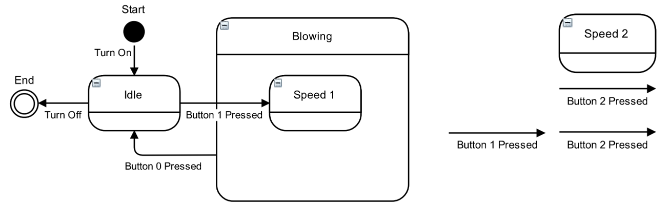
(a) Base model



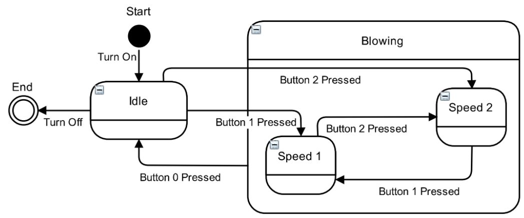
(b) Target model

Fig. 4: The One-Speed Fan example.

2) Two-Speed Fan Activity: The Two-Speed Fan activity (Fig. 5) consumes the model that has been produced in the first activity (Model B in Fig. 2). Therefore, any model generated in the previous activity will become the base model for modelling in this activity. Inline to the Flow concept [11], this second activity should be more challenging. Therefore, the activity (Fig. 5) challenge learners with one additional state and three new transitions. Now the case has changed. The fan has an extra button, Button 2, to support 2-speed blowing. When Button 1 is pressed, the fan blows in speed 1. When Button 2 is pressed, the fan blows in speed 2. Thus, Learners are required modify the Blowing state into a composite state, so it has two-speed states. The fan can move from the Idle state to the Speed 1 state, from the Idle state to the Speed 2 state, and from the Speed 1 state to the Speed 2 state or *vice versa*.



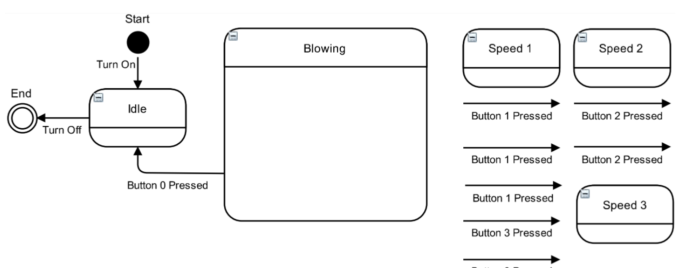
(a) Base model



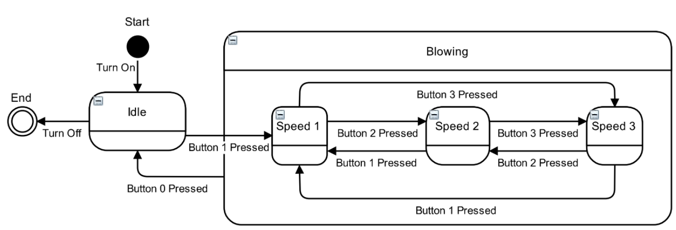
(b) Target model

Fig. 5: The Two-Speed Fan example.

3) Three-Speed Fan Activity: The Three-Speed Fan activity (Fig. 6) should be harder than the second activity. The fan now supports 3 speeds of blowing, but it has been modified so it cannot go directly to Speed 2 and Speed 3 without firstly go the state with lower speed. In other words, the transition from Idle can only go to Speed 1 for the fan to start blowing–Speed 2 and Speed 3 will not be working if transition comes from the Idle state. Thus, starting from the model C as the base model as shown in Fig. 2, learners are required to modify the Blowing state into a composite state, so it has 3 speeds of blowing and only allow transition from Idle to Speed 1 and at a speed that is lower from the intended one to blow.



(a) Base model



(b) Target model

Fig. 6: The Three-Speed Fan example.

D. Generate Game

After finishing constructing the learning pattern (Fig. 2), designers can now generate a path of levels (or stages) that is usually found in many games. Each generated level corresponds to an activity defined in the learning pattern and has an EVL template for validation. The EVL template is shown in Listing 2 which can be extended by designers to write code that fits with the level scenarios and objectives. The number of constraints and operations in the template corresponds to the number of objectives defined in the level's learning pattern activity in Fig. 2. As an example, implementation of validation, Listing 3 shows the EVL code of checking whether Blowing state has already Speed 1 substate as intended by Objective 1 in One-Speed activity.

Listing 2: Validation template for objectives in One-Speed Fan activity/level.

```
context Statechart {
    constraint obj_1 {
        check:
            self.obj_1()
        message:
            "FAIL: ob_1"
    }
    constraint obj_2 {
        check:
            self.obj_2()
        message:
            "FAIL: obj_2"
    }
}
operation Statechart obj_1(): Boolean {
    return true;
}
operation Statechart obj_2(): Boolean {
    return true;
}
```

Listing 3: Validation realisation for Objective 1 in One-Speed Fan activity/level.

```
context Statechart {
    constraint obj_1 {
        check:
            self.obj_1()
        message:
            "FAIL: Blowing state contains Speed 1 substate"
    }
    ...
}
operation Statechart obj_1(): Boolean {
    for (state in State.allInstances.select(state | state.
         name == "Blowing")) {
        if (state.substates.notEmpty() and state.substates.
             select(substate | substate.name == "Speed 1").
             notEmpty()){
            return true;
        }
    }
    return false;
}
...
```

E. Play Game

After generating the learning pattern into a path of ordered levels and defining its levels' validation, Learners can choose the path and play its levels as depicted in Fig. 7. When learners choose the first level, an MxGraph-based editor is displayed. Lesson, model description, instruction, objectives, and base model are presented to learners. Learners then can start to modify the base model to reach the target model.

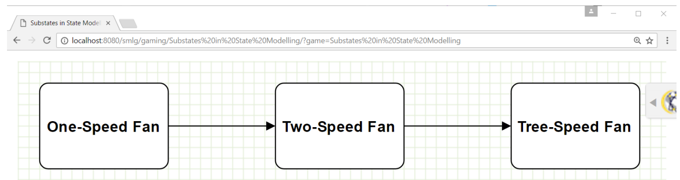


Fig. 7: The path for learning substate concept with its levels.

Every attempt to change the model will trigger an operation to validate the current model using the defined EVL constraints to assess whether the current model has met the current level's objectives. If not, error messages are displayed to learners indicating the objectives that have not been satisfied. For example, in Fig. 8, the Speed 1 substate has not been put into the Blowing state even though the Button 1 Pressed transition has connected the Idle state to the Speed 1 substate. If all objectives have been fulfilled, learners have completed the level. A message that congratulates the learners is displayed to give positive reinforcement. Learners then are brought back to the learning path where they can choose the next level to play.

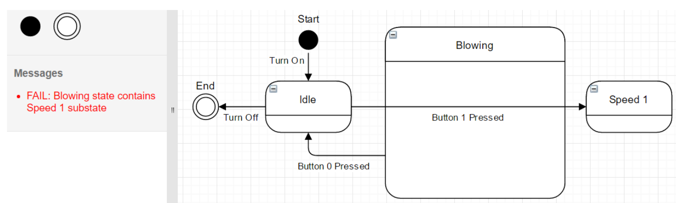


Fig. 8: A message is displayed to learners indicating the objective that has not been fulfilled.

V. Evaluation Plans

Two forms of assessment have been planned to evaluate the framework. The first assessment is to measure the effectiveness of modelling games vs. class-based lecturing using controlled experiments involving undergraduate and postgraduate students. Subjects will be grouped into two groups, one is experimental group, and the other one is controlled, group. While the controlled group will only learn from attending class-based lecturing, the other group will also learn using games. Both will be tested with relevant problems before and after treatment to measure the gap of their improvement, and the gap between the two group is the effect of the games. Even though this evaluation is feasible since an adequate number of software engineering students are available, challenges still exist. The quality of the class-based lecturing, the problems given in the tests, and the games should be consistent, ensuring that they carefully designed to address the same educational objectives. The other challenge is to have the two groups of subjects have a balance quality and distribution.

The second assessment is to measure on the development effort needed to develop a complete game for a non-trivial modelling language, and the savings realised using the model-driven approach. The assessment will be tested on developers that has considerable background in developing games, which also imply a challenge since it's difficult to get developers that have such qualification but also willingly participate as subjects. The measures will be on time used to complete the game and ratio between lines of code written and lines of code generated. Subjective feedback from the developers will also be useful to describe the quality of the framework.

VI. Related Works

There are not many studies that investigate the application of gameful approach in software modelling. So far, only four studies are identified closely relevant to our work. Stikkolorum et al. [7] develop a game that is intended to teach software design principles, such as cohesion, coupling, information hiding, and modularity in object-oriented software design. Groenewegen et al. [8] apply gamification to improve stakeholders' understanding of their enterprise architecture models as well as to validate them. In the domain of information security modelling, Ionita et al.[9] develop a socio-technical modelling language (TREsPASS) that maps information on security-related concepts toward tangible representation. In the context of activity diagram learning, Richardsen [10] develops a game, whose behaviours are controlled through an UML activity diagram. Each of the studies addresses different topics in software modelling using different approaches and game elements. This fact challenges us to develop a more generic approach in addressing software modelling learning so that tutors do not need to develop individual games from scratch only to address certain topics. Moreover, the common drawbacks of the studies are that most of them did not consider the pedagogical aspect of their solution, as well as their validation, was weak in sample size as well as the lack of discussion of internal validity. Nevertheless, all the studies reported that their gamified approaches have a positive effect–it is motivating and engaging users in varying degrees, which confirms that gamification has a positive impact on motivation.

Acknowledgment

This research is part of a doctoral programme funded by *Lembaga Pengelola Dana Pendidikan Indonesia* (Indonesia Endowment Fund for Education).

References

[1] S. Stieglitz, C. Lattemann, S. Robra-Bissantz, R. Zarnekow, and T. Brockmann, *Gamification: Using Game Elements in Serious Contexts*. Springer, 2016.

[2] R. Dörner, S. Göbel, W. Effelsberg, and J. Wiemeyer, *Serious Games: Foundations, Concepts and Practice*. Springer, 2016.

[3] Y. San Chee, *Games-To-Teach or Games-To-Learn: Unlocking the Power of Digital Game-Based Learning Through Performance*. Springer, 2015.

[4] T. M. Connolly, E. A. Boyle, E. MacArthur, T. Hainey, and J. M. Boyle, "A systematic literature review of empirical evidence on computer games and serious games," *Computers & Education*, vol. 59, no. 2, pp. 661–686, 2012.

[5] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work?–a literature review of empirical studies on gamification," in *2014 47th Hawaii International Conference on System Sciences*. IEEE, 2014, pp. 3025–3034.

[6] O. Pedreira, F. García, N. Brisaboa, and M. Piattini, "Gamification in software engineering–a systematic mapping," *Information and Software Technology*, vol. 57, pp. 157–168, 2015.

[7] D. R. Stikkolorum, M. R. Chaudron, and O. de Bruin, "The art of software design, a video game for learning software design principles," *arXiv preprint arXiv:1401.5111*, 2014.

[8] J. Groenewegen, S. Hoppenbrouwers, and E. Proper, "Playing archimate models," in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2010, pp. 182–194.

[9] D. Ionita, R. Wieringa, J.-W. Bullee, and A. Vasenev, "Tangible modelling to elicit domain knowledge: an experiment and focus group," in *International Conference on Conceptual Modeling*. Springer, 2015, pp. 558–565.

[10] O. Richardsen, "Learning modeling languages using strategies from gaming," Master's thesis, Norwegian University of Science and Technology, Norway, 2014.

[11] M. Csikszentmihalyi, *Toward a psychology of optimal experience*. Springer, 2014.

[12] D. Wood, J. S. Bruner, and G. Ross, "The role of tutoring in problem solving," *Journal of child psychology and psychiatry*, vol. 17, no. 2, pp. 89–100, 1976.

[13] L. S. Vygotsky, *Mind in society: The development of higher psychological processes*. Harvard university press, 1978.

[14] S. Deterding, "The lens of intrinsic skill atoms: A method for gameful design," *Human–Computer Interaction*, vol. 30, no. 3-4, pp. 294–335, 2015.

[15] D. R. Krathwohl, "A revision of bloom's taxonomy: An overview," *Theory into practice*, vol. 41, no. 4, pp. 212–218, 2002.

[16] S. Arnab, T. Lim, M. B. Carvalho, F. Bellotti, S. Freitas, S. Louchart, N. Suttie, R. Berta, and A. De Gloria, "Mapping learning and game mechanics for serious games analysis," *British Journal of Educational Technology*, vol. 46, no. 2, pp. 391–411, 2015.

[17] G. Richter, D. R. Raban, and S. Rafaeli, "Studying gamification: the effect of rewards and incentives on motivation," in *Gamification in education and business*. Springer, 2015, pp. 21–46.

[18] R. Ryan and E. Deci, "Self-determination theory: Basic psychological needs in motivation, development, and wellness," 2017.

[19] S. Nicholson, "A recipe for meaningful gamification," in *Gamification in education and business*. Springer, 2015, pp. 1–20.

[20] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, "Eugenia: towards disciplined and automated development of gmf-based graphical model editors," *Software & Systems Modeling*, pp. 1–27, 2015.

[21] D. Kolovos, L. Rose, R. Paige, and A. Garcıa-Domınguez, "The epsilon book," *Structure*, vol. 178, pp. 1–10, 2010.

[22] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[23] D. S. Kolovos, R. F. Paige, and F. A. Polack, "Eclipse development tools for epsilon," in *Eclipse Summit Europe, Eclipse Modeling Symposium*, vol. 20062, 2006, p. 200.