Model-Driven Gamified Software Modelling Learning

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

***Abstract*—In software engineering, software modelling plays a significant role. Nevertheless, learners often consider software modelling as a comparatively difficult subject since it requires them to have abstraction skills to master it. Meanwhile, gamification has been growing as a trend solution to improving learners' engagement. This study endeavours to harness Model-Driven Engineering best practices to construct gamified learning activities that supports learners advancing their modelling abilities. Our method to dealing with the gamified learning combines pedagogical design principles derived from several learning models and the Deterding's Gameful Design framework for it's gamification. Using the Design Science Research Methodology, this research aims to produce a platform for designing and generating gamified software modelling learning. Controlled experiments are planned to be applied to evaluate the effectiveness of the platform as well as the gamified software modelling learning produced.**

***Keywords*—gamified approach, software modelling learning, model-driven engineering**

I. Introduction

[1]

II. Rationales

A. Challenge in Teaching Software Modelling

Software modelling is commonly perceived as a difficult subject since it requires a mastery of abstraction [2]. However, this subject has a fundamental and crucial role in software engineering education and practice. Successful application of software modelling requires skills in abstract modelling [3]. The modelling itself is the process of thinking abstractly about systems [4]. Thus, teaching modelling also means teaching abstraction [5]. Therefore, it is crucial to make students understand the value of abstraction [4]. Weak software modelling skills will likely cause software engineering students to face further challenges with their degrees, as most of the software engineering related subjects involve of inherent abstraction problems [6]. In the context of computer science and software engineering education, Kramer [6] and Hazzan [7] argued that abstraction is the central theme or key skill for computing. Kramer stated, "I believe that abstraction is a key skill for computing. It is essential during requirements engineering to elicit the critical aspects of the environment ... At design time ... Even at the implementation stage ... "[6]. Hazzan also stated, " ... software is an intangible object, and hence, requires highly developed cognitive skills for coping with different levels of abstraction"[7]. The problem of learning appropriate abstraction skills for software modelling is similar to problems in mathematics, where most of the concepts can only be accessed through symbolical representations [8]. Abstraction also requires students to grasp skills in information hiding, generalisation, approximation or reformulation and separating relevant from irrelevant aspects [9]. To overcome these challenges, we need to put effort into software modelling learning design, developing a concrete and motivating presentation which can engage students and facilitate deep learning.

B. Why Gamified Approach?

In recent years, the use of games and game elements for purposes other than leisure has drawn significant attention. Gamification [10] and Serious Games [11] (we use 'gamification' to refer to both concepts) have been proposed as solutions to motivational problems that emerge when users are required to engage in activities that they perceive as boring, irrelevant, or difficult. .

Real-world examples that show the success of the gamification are Duolingo (https://www.duolingo.com) and Re-mission (http://www.re-mission.net/). Duolingo is a gamified system of language learning. It embeds game elements, such as points, levels, and lives, to make language learning more fun. Re-mission is a third-person shooting game dedicated to young cancer patients and designed to teach and learn how to deal with cancer. The patients are invited to take part in an entertaining gameplay that will affect their specific behavioural and psychological outcomes producing effective cancer therapy.

Through a systematic review, Connolly et al. [12] studied the impact of computer games and serious games on engagement and learning in diverse fields. They reported the majority of the studies presented empirical evidence about the positive impact of computer games and serious games. Using the same type of method, Hamari et al. [13] found that according to the majority of the reviewed papers, gamification does generate benefits and positive effects. Specifically in the field of software engineering, Pedreira et al. [14] also performed a systematic review on the application of gamification in software engineering. Most existing studies focus on software development, project management, requirements, and other support areas, but none of them focuses on software modelling.

They also found fewer studies reporting empirical evidence to support gamification research. They argued that existing studies in the field are quite new, thus more research effort is needed to investigate the impact of gamication in software engineering.

Reports of the positive impact of gamification in various fields encourage us to apply it in software modelling learning, an area which has received little attention for the application of gamification so far. This situation broadens our opportunity not only to produce a novel approach to teaching software modelling but also to improve gamification processes through the application of Model-Driven Engineering approaches. This research proposes a main research question, "Can gamification improve software modelling learning?". The word 'improve' implies that learning with gamification enhances learners' engagement and learning performance. The engagement of learners with the support of gamification is more durable, frequent, and active compared to learners that only use didactic approach. Also, the former perform better in knowledge and skill acquisition and application compared to the latter.

C. Related Works

In sections **??** and **??**, we have investigated studies related to abstraction in learning, specifically in software modelling, and best practices in teaching software modelling. From these related studies, we derive several design requirements, which can be found in section **??**. From gamified software modelling related studies in section **??**, each study addresses different topics in software modelling. However, none of them addresses the core topics of Model-Driven Engineering—modelling, meta-modelling, and model transformation, which means that there is an opportunity for novel research in the area. We also found that each study addressed its topic with different approaches and game elements, which also challenges us to develop a more generic design in addressing software modelling learning problems. Moreover, the common drawbacks of the studies are that most of them did not consider the pedagogical aspect of their solution and their validation was weak in sample size as well as the lack of discussion of internal validity. Nevertheless, all of 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.

Still from gamified software modelling related studies in section **??**, we also identified some constructive findings to improve the quality of our research and SMLG design. First, the quality of models created by learners has to be measured to give them feedback how good the models are. Software metrics could be applied to measure the quality. Second, evaluation should conform to the standard criteria of good research practices in terms of sample size and internal validity. Third, pedagogical aspect should be integrated into the SMLG design.

D. Pedagogy's Contribution

Essentially, our study lies in the intersection between software modelling, learning, and games as depicted in Figure **??**. Therefore, pedagogical aspect cannot be neglected. We will apply several learning models from pedagogy to drive the design of our learning game. We also target the SMLG is suitable for higher-level undergraduate and postgraduate students with some of experience of software engineering.

E. MDE Contribution

In the beginning, we plan to address software modelling in Model-Driven Engineering as a whole—comprising modelling, metamodelling, and model transformation. However, after consideration regarding scope and time, we adjust our scoping just to focus on graphical software modelling, which is a common way to express models in modelling and metamodelling. We excluded model transformation since its approaches are commonly expressed in a textual way. We include metamodelling since a metamodel itself is a model of models and usually is expressed in the form of class diagram-like graphics. We plan to perform literature study and develop a prototype in the first year and address modelling and metamodelling in the second year and third year respectively.

Instead of developing the software modelling games manually, we plan to follow a model-based approach. We will develop a design framework for the games, which will systematically and semi-automatically drive gamification design to produce software modelling learning gamification (SMLG). We call the framework the SMLG framework.

F. Potential Contribution

III. Requirements

Through our literature review, we identify some teaching and learning practices of software modelling suggested by experts based on their experience teaching software modelling. The practices are discussed in the following paragraphs.

Modelling is a process to think abstractly about systems. Therefore, modelling is thaught to make students understand the value of abstraction [4]. Additionally, successful application of Model-Driven Software Development depends on abstract modelling skills [3]. Another best practice suggested is that software modelling should be taught with prerequisites [15]. Therefore, the learners should have a good programming background [4] or know a little about OOP [16]. However, for an introduction to modelling in computer science/software engineering programmes, we can teach modelling alongside with programming[17], [4]. In this way, students can learn to model as early as possible [16], [17].

In teaching software modelling, we should encourage students to produce "good" models and measure their "quality", therefore they will be informed how good are their models. One way to measure the quality of models is using tools [16]. In teaching software modelling, problem-solving should be taught first, while modelling language specification and modelling tools can get in the way [15]. Regarding the problem-solving approach, educators should give solutions, not just direct answers [15]. Another best practice is that educators should teach modelling language broadly, not deeply [15], and throughout [17]. Students need to experience the whole cycle of modelling in a software engineering project, so they learn to decide which development process is more appropriate than the others [16]. Consequently, teaching should refer to other disciplines or other aspects related to software modelling as well [15].

Even though code generation is essential to understand modelling, since it shows the real-world application of Model-Driven Engineering, educators need to teach other applications (benefits) of software modelling at first [18]. The code generation could come later [15]. Also, the educators should be careful when using analogies and physical decomposition since they might not reflect the complexity of the system; one component might have a cross-cutting effect to other layers of the system [15]. Furthermore, it is good to teach modelling with a standard language, such as UML [4]. However, educators need to teach modelling and meta-modelling using other modelling languages as well, not just UML. Software modelling is not a UML modelling course [15]. A significant number of successful Model-driven Software Development companies build their own modelling languages and generators, suggesting a re-orientation of education away from UML notation to fundamental modelling principles [3]. Choosing a playful domain or fun problems, not serious domain, is suggested to increase learners' engagement [15].

Throughout the literature review, requirements was gathered. The requirements are the key points pointed out by Model-Driven Engineering (MDE) experts how MDE should be taught and learned. These requirements are useful in the design of learning contents and activities, the pedagogical aspect, of the platform. Table I is a list of requirements summarised from literature review.

From the requirement identification, it is found that the items in the Contents category (Table I) agrees with the item in the Needs category (Table **??**). The finding suggests an agreement about the learning contents that should be delivered to students in learning software modelling.

TABLE I
REQUIREMENTS DERIVED FROM THE LITERATURE REVIEW.

Category	Code	Requirements from Literature Review
Contents	RL01	Teach MDE Definition
	RL02	Teach semantics, syntaxes, notations
	RL03	Teach Modelling, metamodelling, model validation, model transformation
	RL04	Teach the applications of MDE
	RL05	Teach modelling in various domains/contexts
Priciples and Practices	RL06	Modelling is abstract thinking
	RL07	Object-orientation prerequisite
	RL08	Measure student's model's quality
	RL09	Problem solving first, detail specifications and tools get in the way
	RL10	Provide support to solutions, not answers
	RL11	Teach broadly, throughout, not deeply
	RL12	Teach with different modelling languages
	RL13	Make it fun
	RL14	Teach from ground, real-world objects, up to abstraction
Tool Design	RL15	Support and documentation
	RL16	Build knowledge incrementally
	RL17	Flexibility to explore learning
	RL18	Positive reinforcement
	RL19	Convince of the value of the topic being learned
	RL20	High usability

IV. Design

A. Pedagogical Aspect Design

Several existing learning concepts will be applied in the design process of the platform. In this section, we will explain the relationships between the learning models, their contributions, and how they will be applied to the design of the platform are depicted in in Figure 1 and Figure 2.

We decided to implement challenge as the fundamental game element that exists in the design of our game since it is one of the key features that exist in every game. The challenge is a crucial game element since it stimulates and provokes a player to engage with platform. We translate challenge into series of levels in our design and of course higher levels come with higher difficulty. To realise this, we borrow the learning activities—remember, understand, apply, analyse, evaluate, create—from Bloom's taxonomy, since every activity has different cognitive loads according to their order with 'create' has the highest cognitive load. We assume that activity with higher cognitive load is also harder to complete. Therefore, we can make different combinations between the activities that will gradually increase in difficulty (cognitive load) along the increase of the levels (Figure 1). Bloom's taxonomy also act as an inventory of activities that provide us many options of activities that could give variability in our design.

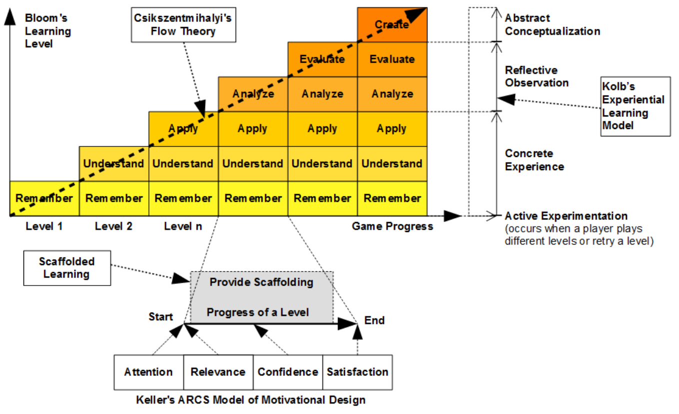

Fig. 1. Elaborating learning models' contribution to the design of the gamified modeling learning

.

While learners are progressing in the platform, they are developing their competence. Thus, difficulty has to be kept balanced with their competence, otherwise they will get bored. It is the situation where the theory of Flow can be applied (Figure 1). To control degree of difficulty, there are three ways we identified so far related to pedagogical approach: a combination of Bloom's activities, the introduction of new

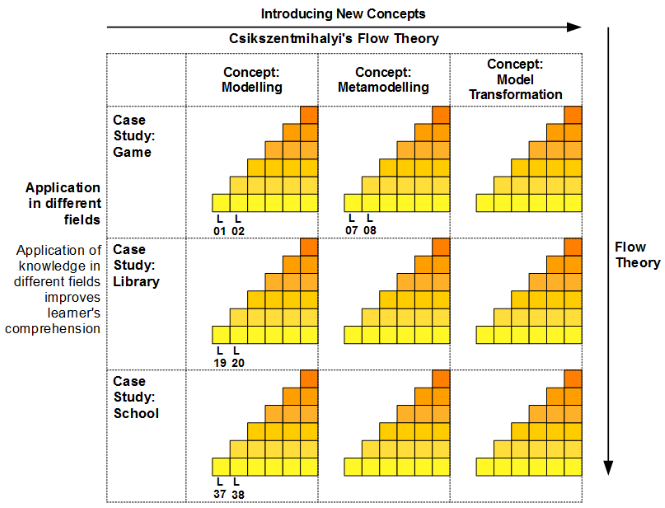
Introducing New Concepts
Csikszentmihalyi's Flow Theory
Concept: Modelling
Concept: Metamodelling
Concept: Model Transformation
Case Study: Game
Case Study: Library
Case Study: School
Application in different fields
Application of knowledge in different fields improves learner's comprehension
Flow Theory
L 01
L 02
L 07
L 08
L 19
L 20
L 37
L 38

Fig. 2. Elaborating learning models' contribution to the design of the gamified modeling learning

concepts, and application in different domains. The order of the levels of each of the three ways has to be arranged properly following the theory of Flow. Concepts that are easier are given earlier than the harder ones, and the difficulty is increased gradually as learners progress. Likewise, Application in the domains that are more familiar with learners should be given first and gradually shifted to the domains that are most unfamiliar (Figure 1. Combining these three dimensions—types of activities, concepts, and domains—could give us a variety of levels with different degrees of difficulties.

Motivation is an important aspect in the success of learning, and we use Keller's ARCS motivational model to address this aspect [19]. The model provides us in each its components—attention, relevance, confidence, satisfaction—a set of predefined techniques to maintain learners' motivation. In a course of a level of platform, there are a start, an end, and learning activities in between (Figure 1). We could apply the ARCS' techniques to maintain learners' motivation along the course of completing a level. As an example, we could use animation to gain learners' attention, explaining the application of the concept being taught in the currently playing level to give relevance, showing their progress in completing a level to maintain their confidence, and giving them a reward after finishing a level for reward.

We apply scaffolding [20], [21] to support learners coping challenges (Figure 1). Throughout finishing a level, scaffolding could be provided in several ways: reducing extensive modelling activities into smaller activity constituents, removing irrelevant activities, providing an almost complete model so they can work on the most relevant activities rather than build the model from scratch, providing help and documentation, and giving some clues of the solutions when they get stuck. This support will be reduced as players progress to maintain the balance between their increasing competence and difficulty.

We also consider applying Kolb's experiential learning model, which is a model that agrees knowledge is constructed through experience and based its model on constructivism [22]. We select Kolb's model since we perceive that playing a level in platform is similar to the learning cycle Kolb proposed; a cycle consists of 4 steps: concrete experience (CE), reflective observation (RO), abstract conceptualisation (AC), and active experimentation (AE).

The cycle can be applied to frame learners' activities in gaining new knowledge through solving a problem given in a level. For example, the first time players play a new level, at that moment they encounters a concrete experience (CE). Immediately, they attempt to identify and characterise the problem given in that level and recall any knowledge that is relevant to solve the problem (RO). Next, they construct a solution for the problem that they face (AC). After constructing the solution, they apply the solution to the problem (AE), experience the result (CE), and then evaluate whether the solution solves the problem of the level or not (RO). Any gap that appears will update their knowledge. They use their newly updated knowledge to produce a new solution (AC) that could be applied to the same problem at the same level or different problem in other levels (AE). AE also occurs when players replay the same level or play a similar level they experienced 'Game Over' or 'Give Up!' decided to choose another level.

TABLE II
REQUIREMENTS DERIVED FROM LEARNING MODELS (SECTION ??).

Category	Code	Requirements from Learning Models
Learning Models	RM01	Design satisfies Bloom's taxonomy.
	RM02	Design suffices Kolb's experiential learning model.
	RM03	Design meets Keller's ARCS motivational model.
	RM04	Design fulfils scaffolded learning.
	RM05	Design complies with the theory of Flow.

Learning activities in Bloom's taxonomy also correspond to the steps in Kolb's learning cycle [23] and both have been applied together to design instructions in different fields [24], [25], [26]. Therefore, we argued that both could be implemented simultaneously; Bloom's taxonomy provides learning activities while Kolb's model addresses learning cycles in the design of our game. To simplify our work, we summarise the elaboration of design and learning models into a list of requirements (Table II) that will be used in the design and evaluation activities.

B. Game Aspect Design

In this research, we plan to assess whether gamification is beneficial for learners of graphical software modelling languages. We choose graphical modelling languages since they are the common languages used in modelling, whether in academia or industry, and extensively used in Model-Driven Engineering. Standard graphical modelling languages like UML(http://www.uml.org) and BPMN(http://www.bpmn.org), often used in Model-Driven Engineering, are some of the use-cases.

Modelling can be expressed in different modelling languages. To minimise bias and ensure the generality of our platform, we plan to experiment and support several graphical modelling languages (e.g. UML, BPMN, state-charts). For each modelling language, we envision the development of dedicated platform that will be derived from the Gameful Design Framework [27]. The platform will mimic a graphical modelling tool, and at each level, it will require the learner to graphically construct or adapt a model to meet a set of constraints and requirements.

The platform will have levels of gradually increasing difficulty as well as variety in its challenges, to expose learners to different kinds of domains, models, and diagrams. Tutorials are planned to be embedded into the platform to help learners familiarise themselves with the control system and the flow of the platform.

The platform will incorporate interim goals and intrinsic rewards to motivate learners. Each type of modelling language (e.g. object modelling, collaboration, process) will have several stories. A story will represent a specific case study to introduce learners to particular problems in specific domains. Every story will be composed of several levels, and every level will have one or more objectives that a learner needs to accomplish to complete it. A level may also be a continuation of a previous level, giving the learner step-by-step progression to complete the domain problems. Each story and level will introduce new concepts and link them with previously introduced concepts.

A real-world problem can be time-consuming and very complex to model. Thus, the inessential activities that are not significant to the core concepts that are being taught should be excluded. As a result, learners will be more focused on the main concepts. Thus, game elements like limited choices (i.e. only limited items can be dragged), microflows (i.e. put the right element to its right place), and bite-sized actions (e.g. drag and drop) will be implemented to facilitate learners in performing the core activities. Likewise, fuzziness will also be used to stimulate learners' creativity since most of the time there is no single correct model for the problem at hand. Attractive design will also be significant to motivate learners to interact with the platform. The platform should be able to give instant, noticeable, and actionable feedback to maintain learners' engagement and monitor their progress. Interesting and varied feedback should be designed to appeal to the learners' motives. We also plan to implement the platform using web technologies so that they are accessible to a wide audience.

The details of the application of Deterding's Gameful Design to our design process are presented in Appendix **??**. The process also produced storyboards that are the preliminary design of levels and graphical user interface of our game (Appendix **??**).

C. Model-Driven Aspect Design

We plan to build a platform that will facilitate the design and generation of platform. Rather than developing platform for each graphical modelling language manually, we will follow a model-based approach. In the spirit of Eugenia [28], we will use metamodel annotations to define the graphical syntaxes of modelling languages and separate models to specify the game elements (constraints, objectives, levels, etc.) of the platform. These models will be then consumed by a model-to-text transformation to produce fully-functional language-specific platform. Thus, the platform supports software modelling tutors in the design and customisation of the platform at the high level of abstraction as well as to automatically build the platform.

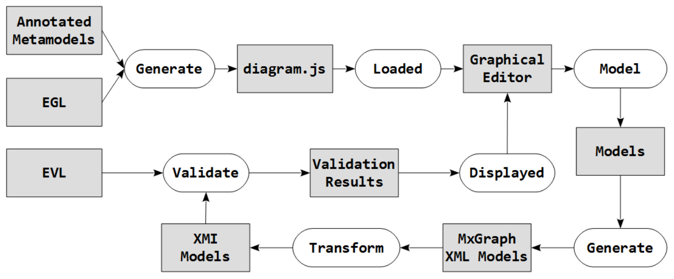

Fig. 3. The process how the platform works from generation of visual modelling notations, modelling on graphical editor, to validation of models.

V. Demonstration

VI. Conclusion

Acknowledgment

VII. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] R. H. Von Alan, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[2] J. Börstler, L. Kuzniarz, C. Alphonce, W. B. Sanders, and M. Smialek, "Teaching software modeling in computing curricula," in *Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups*. ACM, 2012, pp. 39–50.

[3] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial adoption of model-driven engineering: Are the tools really the problem?" in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2013, pp. 1–17.

[4] J. Bezivin, R. France, M. Gogolla, O. Haugen, G. Taentzer, and D. Varro, "Teaching modeling: why, when, what?" in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2009, pp. 55–62.

[5] G. Engels, J. H. Hausmann, M. Lohmann, and S. Sauer, "Teaching uml is teaching software engineering is teaching abstraction," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2005, pp. 306–319.

[6] J. Kramer, "Is abstraction the key to computing?" *Communications of the ACM*, vol. 50, no. 4, pp. 36–42, 2007.

[7] O. Hazzan, "Reflections on teaching abstraction and other soft ideas," *ACM SIGCSE Bulletin*, vol. 40, no. 2, pp. 40–43, 2008.

[8] R. Duval, "A cognitive analysis of problems of comprehension in a learning of mathematics," *Educational studies in mathematics*, vol. 61, no. 1-2, pp. 103–131, 2006.

[9] L. Saitta and J.-D. Zucker, *Abstraction in artificial intelligence and complex systems*. Springer, 2013, vol. 456.

[10] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, “From game design elements to gamefulness: defining gamification,” in *Proceedings of the 15th international academic MindTrek conference*. ACM, 2011, pp. 9–15.

[11] D. R. Michael and S. L. Chen, *Serious games: Games that educate, train, and inform*. Muska & Lipman/Premier-Trade, 2005.

[12] 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.

[13] 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.

[14] 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.

[15] R. F. Paige, F. A. Polack, D. S. Kolovos, L. M. Rose, N. Matragkas, and J. R. Williams, “Bad modelling teaching practices,” in *Proceedings of the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS14), Valencia, Spain*, 2014.

[16] S. Akayama, B. Demuth, T. C. Lethbridge, M. Scholz, P. Stevens, and D. R. Stikkolorum, “Tool use in software modelling education.” in *EduSymp@ MoDELS*, 2013.

[17] J. Börstler, L. Kuzniarz, C. Alphonce, W. B. Sanders, and M. Smialek, “Teaching software modeling in computing curricula,” in *Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups*. ACM, 2012, pp. 39–50.

[18] G. Liebel, R. Heldal, J.-P. Steghöfer, and M. R. Chaudron, “Ready for prime time,-yes, industrial-grade modelling tools can be used in education,” *Research Reports in Software Engineering and Management No. 2015:01*, 2015.

[19] J. M. Keller, *Motivational design for learning and performance: The ARCS model approach*. Springer Science & Business Media, 2010.

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

[21] 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.

[22] D. A. Kolb, *Experiential learning: Experience as the source of learning and development*. FT press, 2014.

[23] E. J. Murphy, “Prior learning assessment: A review of bloom’s taxonomy and kolb’s theory of experiential learning: Practical uses for prior learning assessment,” *The Journal of Continuing Higher Education*, vol. 55, no. 3, pp. 64–66, 2007.

[24] R. E. Terry and J. N. Harb, “Kolb, bloom, creativity, and engineering design,” in *ASEE Annual Conference Proceedings*, vol. 2, 1993, pp. 1594–1600.

[25] R. A. Howard, C. A. Carver, and W. D. Lane, “Felder’s learning styles, bloom’s taxonomy, and the kolb learning cycle: tying it all together in the cs2 course,” in *ACM SIGCSE Bulletin*, vol. 28, no. 1. ACM, 1996, pp. 227–231.

[26] L. Schatzberg, “Applying bloom’s and kolb’s theories to teaching systems analysis and design,” in *The Proceedings of ISECON*, vol. 19, 2002.

[27] 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.

[28] 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.