Qualifying Dissertation

Gamified Software Modelling Learning

Alfa Ryano Yohannis
ary506@york.ac.uk

Supervisor:
Dimitris Kolovos
Fiona Polack

Department of Computer Science
University of York
United Kingdom

October 17, 2016

Abstract

Software modelling has a fundamental role in software engineering. However, it is perceived as relatively challenging for learners to develop the necessary abstraction skills to master the subject. On the other side, gamification is now flourishing as a popular strategy to engage learners. This research attempts to exploit gameful design as an innovative approach, used to create games that reinforce learners' mastery of software modelling by developing their abstraction skills. Our approach to gameful design brings together gamification development concepts such as the Lens of Intrinsic Skill Atoms, and pedagogical design principles from several learning theories and models. The research follows the Design Science Research Methodology and exploits Model-Driven Engineering best practices. The target outputs of this research are a modelling game design and generation framework, and a number of games produced using it. The effectiveness of the framework and its games will be evaluated using controlled experiments.

Contents

Chapter 1

Introduction

This research is motivated by the growing attention towards the application of gamification in various fields [1]. Gamification itself is defined as transforming an entity to become more gameful by embedding game-associated characteristics into the entity without losing its original intended functions. The common goal of gamification is improving users engagement with the entity through motivating by making the entity more fun. We perceive this as an opportunity to apply gamification as well to software modelling courses. Software modelling is a subject that is seen challenging to students since it requires lots of abstraction. Thus, students who didnt possess this kind of ability might be hindered and demotivated. Employing gamification is expected will support the students to maintain their engagement along the courses and, therefore, improve their learning outcomes.

However, in applying the gamification, we faced a challenge that there are only several papers available that discuss the application of game characteristics in software modelling and most of them are still in the preliminary research, covering only small specific topics of software modelling. Modelling, metamodelling, and model management, the core concepts of model-driven software engineering, havent been covered yet. Therefore, we believe that there is a need to extend existing gamification of software modelling works to cover those core concepts as well. The gamification itself is expected can also extend our existing approaches in teaching and learning software modelling. The purpose of this research is to study the gamification of software modelling with emphasis on (1) to what extend gamification improves the learning outcomes of software modelling learning, (2) understanding the process of how gamification improves the learning outcomes, and (3) the design of gamification of software modelling learning manifested in the form of gamification frameworks.

We explain the literature review in Section 2 and propose the research proposal in Section 3. In Section 4, we describe the preliminary results. Finally, we end this qualifying dissertation with conclusions in Section 5.

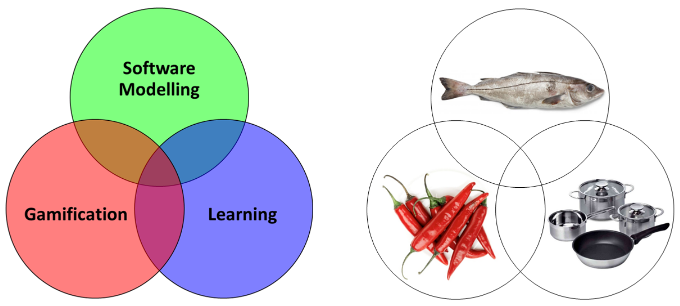


Figure 1.1: How to 'Cook' a Gameful Software Modelling Learning?

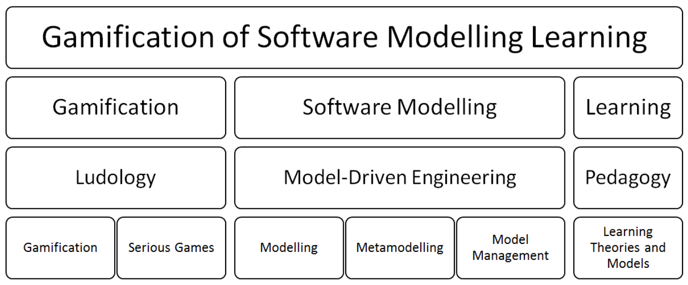


Figure 1.2: Topics the gamification of software modelling learning.

Chapter 2

Field Survey and Review

Since the research of the gamification of software modelling learning is multidisciplinary, its best to organise the literature review categorised by the contributing fields, including the subtopics resulting from the interaction between them. So far, we have identified three major contributing fields, the software modelling itself, ludology, and pedagogy as well. They will be discussed in the following subsections and a brief review about the research methodology ends this chapter.

2.1 Pedagogy

Designing the gamification of software modelling learning cannot be separated from the field of pedagogy since the core process that will be supported by the gamification is the learning process itself. Pedagogy is a source of knowledge of how we can understand learning processes and drawn principles that will guide the design of the artefact of this research. We have selected several theories and models. They are discussed briefly in the following subsections.

2.1.1 Csikszentmihalyi's Flow Theory

Csikszentmihalyi proposed Flow Theory which states that in order to maintain one's engagement in an activity, the balance between challenges and his capability has to be maintained [2]. Once the challenges are too difficult, he arrives in a state of anxiety that demotivating him and then force him to withdraw from the activity. On the other side, if the challenges is too easy for him or his skills are very advance, this condition will make him into the state of boredom which also force him not to continue his engagement in the activity.

This theory has been applied in many fields, such as education, training, and games. Specifically for games, flow theory influences the design of levels. The given challenges have to adjusted so they are balance to the skills of players that improve along playing with games. In the same way, this theory will also influence the design of our game.

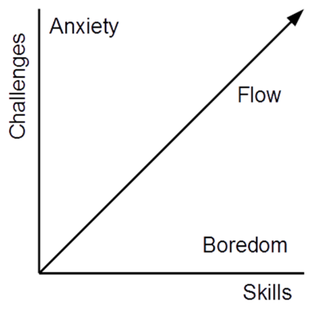


Figure 2.1: Csikszentmihalyi's Flow Theory [2].

2.1.2 Zone of Proximal Development/Scaffolding

Zone of Proximal Develoment (ZPD) was proposed by Lev Vygotsky in the context of adolescent development [3]. This term was also called Scaffolding by Wood et al. in the context of learning [4]. This theory states that during a learning process, learners have to be reinforced, particularly for knowledge or skills that are too difficult to learn without the support of others, to develop their competence until they are able to acquire the knowledge and skills on their own.

Software modelling learning game should be designed according to the principle. It should implement scaffolding, providing helpers and cues to learners to support them solving problems. The scaffolding the will be removed gradually as the competence of the learners grows.

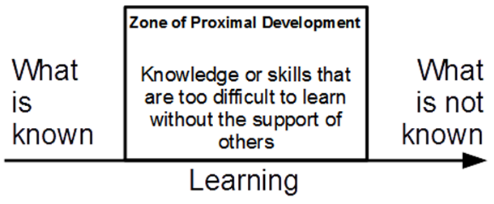

Zone of Proximal Development
What is known
Knowledge or skills that are too difficult to learn without the support of others
What is not known
Learning

Figure 2.2: Zone of Proximal Development/Scaffolding [3],[4].

2.1.3 Revised Bloom's Taxonomy

Bloom's Taxonomy [5] is a framework of cognitive levels for learning process. The framework has been proven very useful for more than 50 years, all over the world, in supporting educators design learning instructions [6]. It consists of six activities for students in learning, namely remember, understand, apply, analyse, evaluate, and create, and order them according their cognitive load levels with 'remember' at the bottom and 'create' as the activity that require cognitive load most.

We see the potential of using the Bloom's Taxonomy activities to design the activities and challenges of learners when playing software modelling learning game. The six activities are the activities that the learners are going to perform when playing the game and the nature of cognitive load of each activity is the challenges of the game. The Bloom's Taxonomy gives us variety of options, activities and challenges, in designing of the levels of our game.

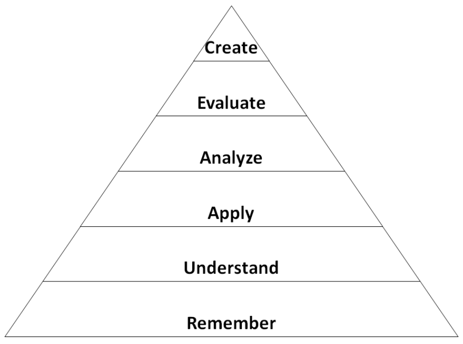


Figure 2.3: Revised Bloom's Taxonomy[5].

2.1.4 Kolbs Experiential Learning Model

Kolb's Experiential Learning Model is a model of teaching and learning through experience and reflection on actions [7]. It states that knowledge development is a product of experience or iterative search. This model implies that software modelling learning game should allow learners to perform active experimentation and develop their own understanding.

The model is a cycle of four steps: abstraction conceptualisation, active experimentation, concrete experience, and reflective observation. In the case of didactic learning with practicals, learners are firstly taught about the theory. In this step learners perform abstract conceptualisation. They then move to the practicals which they are asked to perform some active experimentation of the theory they got in the class. Through the experimentation they feel the the concrete experience. The results might be to some degree confirm to or disagree with the theory. In the end, through reflective observation, the results are then reconciled with the knowledge they had built based on the theory, whether the knowledge is updated or not.

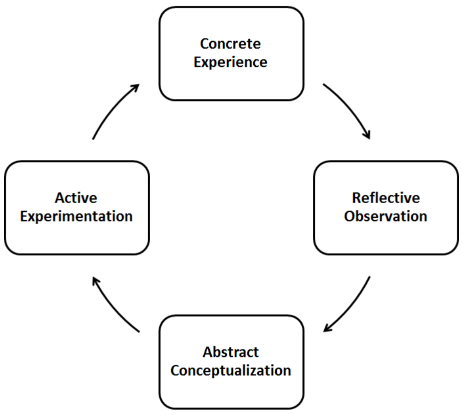
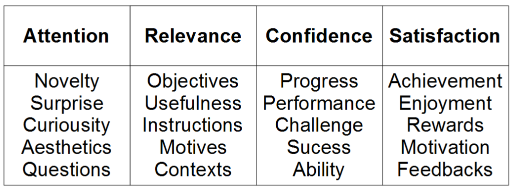


Figure 2.4: Kolbs Experiential Learning Model [7].

2.1.5 Kellers ARCS Motivational Learning Model

Keller's Motivational Model is a set of steps for promoting and maintaining motivation in a learning process [8]. The steps are attention, relevance, confidence, and satisfaction. This model is useful for designing the flow of a stage or level of a software modelling learning game as it mentions steps that we need to focus in order to maintain learners' motivation.

To maintain the motivation of learners in a process of learning, first, we need to draw the attention of learners using novelty, surprise, aesthetics, or questions. After that, we should explain the relevance of the topic that is about to be explained. We could present the objectives, usefulness, instructions, motives, and contexts. Along the learning process, learners' motivation is maintained by raising the confidence of the learners through presenting their progress, performance, challenge, success, and ability. In the end of the learning process, learners should experience the satisfaction which can be realised by presenting them achievement, enjoyment, rewards, motivation, and feedback.



Attention	Relevance	Confidence	Satisfaction
Novelty Surprise Curiousity Aesthetics Questions	Objectives Usefulness Instructions Motives Contexts	Progress Performance Challenge Sucess Ability	Achievement Enjoyment Rewards Motivation Feedbacks

Figure 2.5: Keller's Motivational Model [8].

2.2 Ludology

2.2.1 Serious Games and Gamification

2.2.2 Hunicke's MDA Model

Hunicke's MDA Model [9] consists of three steps in designing a game. They are menchanics, dynamics, and aesthetics. Mechanics means the algorithms and data representation used as the backend of the game. Dynamics is runtime behaviour, mechanics, and player inputs and outputs overtime. It is the interaction between players and the game. Aesthetics is player's emotional responses. Design of a game has to balance challenges and player's skills to keep the player flowing.

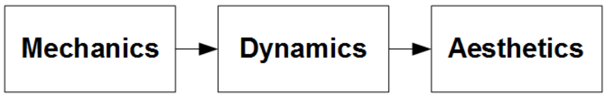


Figure 2.6: Hunicke's MDA Model [9].

2.2.3 Werbach's Gamification Framework

Werbach and Hunter proposed a Gamification Framework to guide the implementation of a gamification [10]. The framework consist of a prescriptive model of implementing gamification and a structural model of game element hierachy. The prescriptive model consists of 6D implementation steps, which are (1) Define business objectives. (2) Delineate target behaviours. (3) Describe your players. (4) Devise activity cycles. (5) Don't forget the fun! (6) Deploy the appropriate tools.

The hierarchy consists of three layers, dynamics, mechanics, and components. Dynamics are the big-picture aspects of the gamified system that one needs to consider and manage but which can never directly present in the system. Mechanics are the basic processes that drive the action forward and generate player engagement. Component are the specific instantiations of mechanics and dynamics.

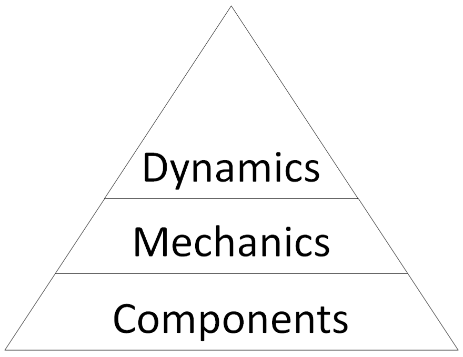


Figure 2.7: Werbach's game element hierarchy [10].

2.2.4 Learning Mechanics-Game Mechanics Model

Learning Mechanics-Game Mechanics (LMGM) Model consists of 2 sets of mechanics: learning mechanics (LM) and game menchanics (GM) [11]. Each mechanics comprises of activities relevant to the aspect that each represents. For examples, modelling, analyse, and discover in LM, and role play, strategy/planning, and selecting/collecting in GM. They act like an inventory which a game designer can select the best mechanics from each mechanics group that fit to a context. The mechanics are then integrated into players' activity cycles.

2.3 Software Modelling

2.3.1 What is Software Modelling?

concretisation

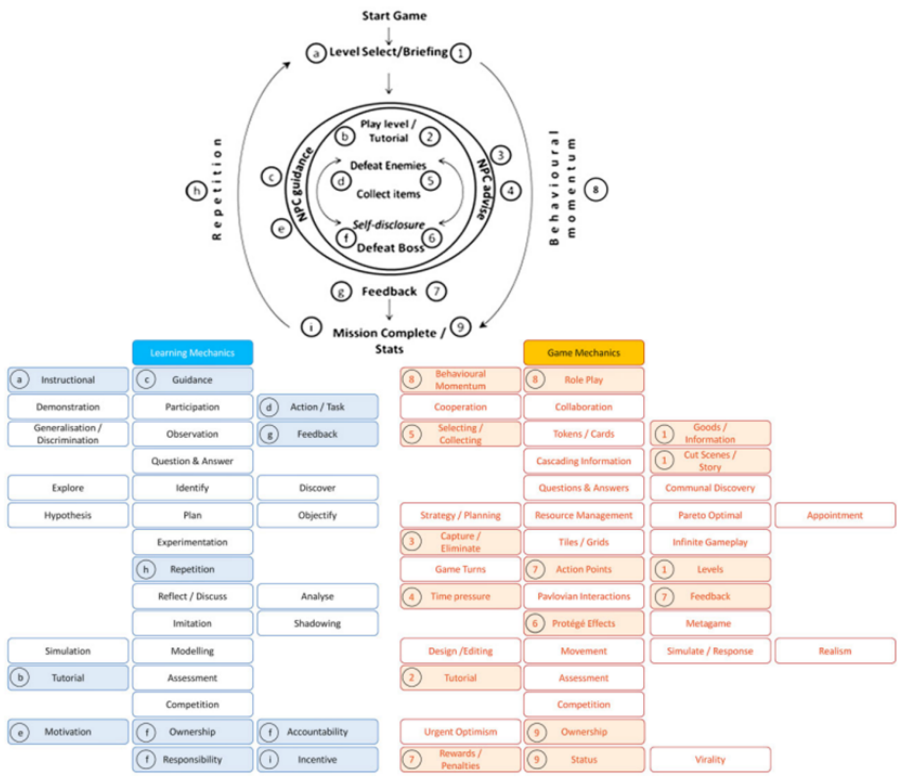


Figure 2.8: Learning Mechanics-Game Mechanics Model [11].

2.3.2 Software Modelling Tools

2.3.3 Software Modelling Teaching

We have investigated related literatures regarding the teaching and learning of software modelling. From the literatures, we have identified lessons and categorised them into 3 groups: contents, teaching or learning practices, and tool design. We will take into account the lessons during the design process of our gamified software modelling learning since we want to develop our research upon other existing works thus have stronger foundation.

Contents

Contents mean the software modelling topics and their structures. In teaching software modelling, we need to teach the core, important concepts and their relation with the contexts and applications of the outside world. The contents of software modelling learning should be:

1. Software modelling definition [13].

2. Semantics, syntaxes, notations. Teach modelling foundations, but focus more on semantics, not only syntax [13]. Improve use of language, not only vocabulary [14].

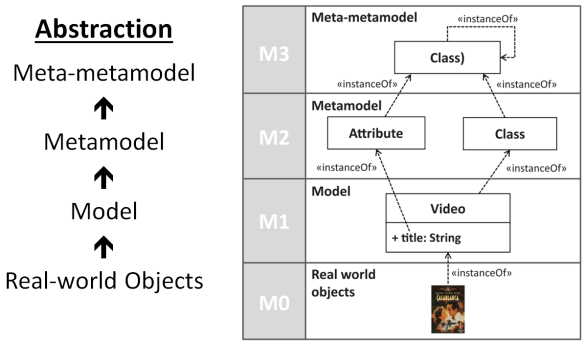


Figure 2.9: Abstraction in Model-driven Engineering [12].

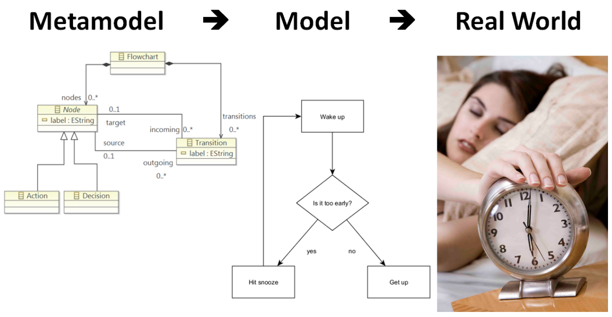


Figure 2.10: Instantiation in Model-driven Engineering.

3. Modelling, metamodelling, and model transformation. Focus on the core, important concepts: modelling, meta-modelling, and model transformation. Teach modelling that comprise the following topics: formal and informal models, partial and complete models, distinction between models and programs [14]. Meta-modelling is given a larger portion than modelling since a model might conforms to more than one meta-mode [15].

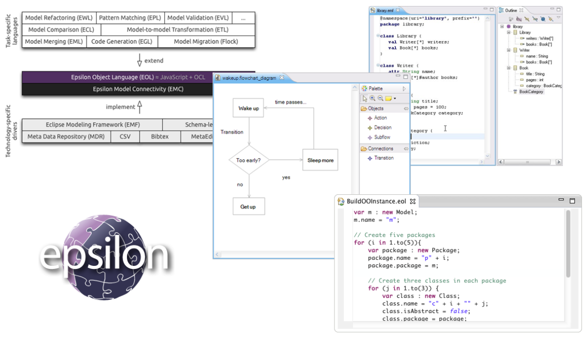
4. Software modelling is engineering. Teach the engineering aspects of software modelling: understanding a domain, planning and resourcing, documentation, quality, formality, validity, optimisation [16].

5. Contexts and practices/applications in various domains [16]. Teach the application of software modelling in various domains, success stories, code generations, model discovery, and model-driven interoperability. Make the model executable. Still the power of execution makes it much easier to understand the model [14]. Convey the practical applications of modelling [13].

Teaching and Learning Practices

Teaching and learning practices are the principles, values, and methods in teaching or learning software modelling. The followings are them:

1. Modelling is the process to think abstractly about systems. Therefore, we teach modelling to make students understand the value of abstraction [14].



```
@namespace(uri="library", prefix="")
package library;

class Library {
   val Writer[*] writers;
   val Book[*] books;
}

class Writer {
   attr String name;
```

```
var m : new Model;
m.name = "m";

// Create five packages
for (i in 1.to(5)){
    var package : new Package;
    package.name = "p" + i;
    package.package = m;

    // Create three classes in each package
    for (j in 1.to(3)) {
        var class : new Class;
        class.name = "c" + i + "" + j;
        class.isAbstract = false;
        class.package = package;
```

Figure 2.11: Epsilon is a family of languages and tools for model management[1].

Successful application of Model-Driven Software Development requires skills in abstract modelling [17].

2. Teach with prerequisites [16]. Student should have a good programming background [14] or has to know a little about OOP [18].

3. Encourage students to produce "good" models, and measure the "quality". One way is to use tools [18].

4. Learn modelling as early as possible [18], [13]. Teach modelling together with programming (Brstler et al., 2012). Modelling should be developed alongside programming [14].

5. Problem-solving first, modelling language specification and modelling tools get in the way [16].

6. Provide solutions, not answers [16].

7. Teach modelling language broadly, not deeply [16], and throughoutly [13]). 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 [18].

8. Refer to other disciplines or other aspects related to software modelling [16].

9. Even though code generation is essential to understand modelling [19], teach other applications (benefits) of software modelling at first. Code generation comes later [16].

10. 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 [16].

11. It is good to teach modelling with a standard language, such as UML [14], but teach modelling and meta-modelling using other modelling languages as well, not just UML. Software modelling is not a UML modelling course [16]. A significant number of successful Model-driven Software Development companies

build their own modeling languages and generators, suggesting a re-orientation of education away from UML notation to fundamental modeling principles [17].

12. Choose a playful domain or fun problems, not serious domain [16].

Tool Design

Tools design are the principles in designing software modelling tools. Since we are going to develop a tool that can support learners to learn software modelling, lessons in this category will have important in guiding the design of our tool.

1. Learning modelling tools is not trivial (Paige et al., 2014).
2. Build knowledge and skills incrementally [20].
3. Use papers, tools designed for pedagogy, or use mainstream tools [18].
4. The usage of modelling tools is not important in the beginning, but later when modelling task becomes larger [14].
5. A good experience with the tool has a positive inuence on the students view of UML and modelling in general [19].
6. Support for tool usage greatly affects student satisfaction. Provide Tool Expert and carefully design the instructions if you are using Industry Modelling Tool [19]. For an example, Papyrus is suitable for a classroom environment if given the right level of support [19].
7. Give positive reinforcement to learners [20].
8. The tools gives maximum opportunities for learning [20].
9. The tool convinces learners of the value of what they learn [20].
10. Focus on high usability [20].
11. The tool is inexpensive [20].

2.3.4 Abstraction in Software Modelling Learning

Following the last discussion, modelling and meta-modelling, and additionally model management, are the potential concepts that will be taught in the game. All the three concepts requires adequate abstraction skill; a skill that has a fundamental role in Computer Science and Software Engineering (Engels, Hausmann, Lohmann, & Sauer, 2006; Hazzan, 2008; Kramer & Hazzan, 2006). There are some available strategies in teaching abstraction. Familiarity, Similarity, Reification, Application. This strategy is proposed by White and Mitchelmore in the Math education (White & Mitchelmore, 2010). They argued that abstraction should be developed from empirical experience, move up to abstraction, and concreting the abstraction through application. Similarly, based on their experience teaching UML, Engels et al. (Engels et al., 2006) approached abstraction through modelling according to this steps: real-word objects, object diagrams, class diagrams. The real-world objects can be described by videos or may be substituted by textual explanation, animations, or pictures. Hazzan (Hazzan, 2008) proposed three methods in teaching abstraction. First, illustrate. Lecturer uses abstraction-related words or statements in teaching. Second, reflect. For an example, lecturer and students question the impact of using certain level of abstraction or not using abstraction at all. Third, practice. Students must practice and reflect on what he do.

There are two approaches of learning software modelling which a learner may perform them simultaneously: rational approach and intuitive approach (Bobkowska, 2014). Usually at the beginning of modelling process, one uses his existing knowledge and logical reasoning to understand and develop software models, and sometimes preceded with empirical activities. However, there are situations in software modelling when his existing knowledge and methodology-based, logical reasoning are not enough to make him understand or to develop a software model. Thus, make him relies on his own intuition or both approaches interplay in dealing with complex, difficult modelling. In line with Bobkowska's statement, in building a model on the correct level of abstraction, one needs a certain intuition and skill which cannot be gain through lectures but have to be experienced through exercises (Engels et al., 2006).

2.3.5 Gamified Software Modelling

Most of the gamification studies available are dominantly relates to software engineering in a larger context or other aspects of software engineering, such as software implementation and project management, rather than software modelling in particular [21]. Several studies that apply gamification specifically for software modelling are the works of Stikkolorum et al. [22], Ionita et al. [23], Groenewegen et al. [24], and Richardsen [25]. These works are selected because they develop artefacts based on gamification approach, apply them to software modelling, and validate the results.

Puzzle Game to Teach Software Design Principles

Stikkolorum et al. [22] developed a game that is intended to teach software design principles, such as cohesion, coupling, information hiding, and modularity in software design.They tried to look for a solution that provides balance between coupling and cohesion by using the toolbox to draw classes, methods, attributes, and relationships between them. They applied game elements like puzzle game, game levels, visual and audio feedbacks, progress indicator, level unlocking, choice of path, multiple solutions, scoring. For the pedagogical aspect, they mentioned Blooms taxonomy in their work. However, they did not explain how they integrated the taxonomy into their work. After using their game, users started to talk regarding classes, methods, and associations instead of boxes, blocks, and lines, indication unconscious learning. The challenges of their work were on determining scores because there are more than one solutions for a problem. To determine coupling, they used Coupling Between Classes (CBO). Cohesion was measured by comparing all itemsattributes, methods, class namein a class that have similar keywords. Information hiding and modularity is evaluated using general design patterns. They validate their design by conducting user test and utilise the 'think aloud' methodasking users to tell their thoughts while using the game.

Explorable Board-game to Understand and Validate Enterprise Architecture

Groenewegen et al. [24] applied gamification to improve stakeholders' understanding of their enterprise architecture models as well as to validate them. They employ exploring the model step by step, element by element according to the given rules technique which can provides a player a progressive user experience. Therefore, it can improve user understanding. The game proposed is more playable, more freedom to try and explore, and no explicit rewards given. For the game elements, they utilise cards, explorable board-game, and rules. Regardless of their claim that users can understand the model better rather than by merely looking at the model so they can give argument whether the model is valid or not based on their existing knowledge, they never clearly discussed the pedagogical aspect of their work. The challenges that they experienced during the implementation of their are preparing the game by translating implicit knowledge to explicit knowledge of the modela gap of knowledge between the modeller and the reader, which domain knowledge is required. Lack of domain knowledge will make the model less understandable and the user cannot validate the model. For validation, they tested their work to seven respondents and then interviewing them.

Familiar Tangible Model to Model Information Security

In the domain of information security modelling, Ionita et al.[23] developed a socio-technical modelling language (TREsPASS) and map them toward tangible representation. Mapping the socio-technical modelling language to the tangible model is the most challenging part of their work. The tangible representations will increase the familiarity and understandability of models, which will increase awareness, involvement, and learnability. For the game elements, they utilise familiar, tangible representation, such as Lego characters, board-game metaphor, rules. They also discussed the pedagogical aspect of their design which is based on the theory of constructionvism, cognitive load, and cognitive fit. Moreover, based on their experiment, they reported that experimental group performs better than the control group in learnability, efficiency, correctness, and satisfaction. Likewise, based on interview with experts and professionals, the respondents argued that the tangible model might be useful for less technical domain experts and different types of stakeholders to be more participative and contributive in the early stages of architecture modelling.

Arranging UML Activity Diagram to Control the Behavior of a Game

In the context of activity diagram learning, Richardsen [25] developed a game which it's behaviours are controlled through arranging UML activity diagram. Throughout his research, he found challenge that controlling game from activity diagram in Reactive Block Environment (Eclipse based) is difficult since Eclipse is difficult for a first-time user. For validation, he conducted user testing with three users. Think Aloud method was used for observation. After that, questionnaires were given and an interview was conducted. he found out There was no significant different between the traditional interactive tutorial and the game-like tutorial on their performances. However, the game-like tutorial was more engaging. For the validation of his work, he did not mention any explicit pedagogical aspect.

Based on the previous four related works, we concluded that different concepts of software modelling were addressed. However, no use of game elements that addresses abstraction regarding modelling, meta-modelling, and model transformation in particular, which mean there is an opportunity for research on that area. We also found that different specific concepts were addressed with different approaches and game elements which also challenge us to develop a more generic design in addressing software modelling learning problem. The good news is all of the works reported that the use of game and game elements has positive effect in motivating and engaging users in varying degree. Furthermore, every study has their own challenges which are a good thing to be aware of when carrying our research. The drawbacks of other works are, first, most of the them did not consider seriously about pedagogical aspect of their solution and, second,in general, their validation was weak regarding sample size and lack discussion of internal validity.

2.3.6 Software Modelling Course at the University of York

2.4 Research Methodology

The software modelling learning game design framework will be derived deductively mainly from existing related theories and research works that come from the fields of education, games, and software modelling. The resulting framework will be tested in several case studies for validation and the test results will be inputs to revise the framework. Since the outputs of the research will be design artefacts, Design Science Research Methodology [26] is selected as the research methods as it provides a comprehensive conceptual framework and activity guidelines for understanding, developing, executing, and evaluating design artefacts. We also implements Deterding's Gameful Design Steps [27] to guide the design of the gameful aspect of the the product.

2.4.1 Design Science Research Methodology

We will employ Design Science Research Methodology [26] as the methodology to carry out the research. DSRM is selected since it provides a comprehensive conceptual framework consists of activity guidelines for understanding, developing, executing, and evaluating design artefacts. Oher reason is that it positions itself at the top level of abstraction without going into much detail of how to perform each activities, we can freely choose other more concrete research methods to carry out the activities. For examples, we can conduct literature reviews, surveys, or expert interviews to determine research problems, motivations, solutions, and objectives as well as controlled experiments to measure and evaluate the effectiveness of the artefacts.

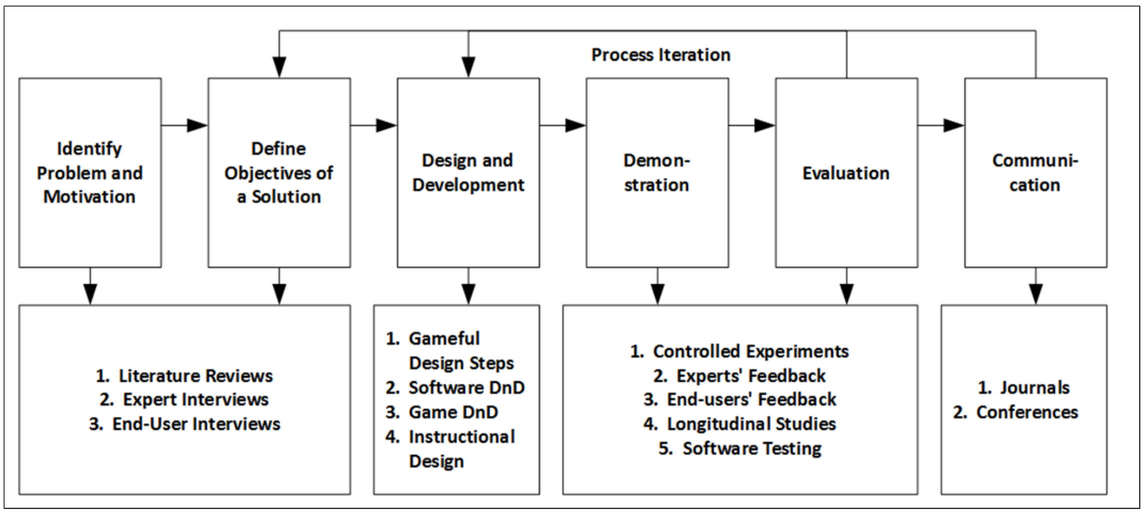


Figure 2.12: Design Science Research Methodology. Adapted from Peffer et al. [26].

Identify Problem and Motivation. This research will use literature review, suggestions from experts, and surveys to identify research problems and motivations a well as to determine the solution and its objectives.

Based on the literature review, the best guesses that we have made so far regarding the challenges that student has to deal with when learning software modelling are abstracting (choosing the most relevant elements, ignoring the irrelevant ones, and determining the relationships between elements), diagramming (translating the abstraction into visual elements), and determining the algorithmic strategies (programming in the context of model management). Other challenges are no fun (the presentation of the software modelling course is not interesting) and the heavy cognitive loads when dealing with abstract objects. However, the guesses have to be confirmed yet through surveys, drawing information from students of software modelling courses in order for our problem identification to be more accurate.

As we could view the challenges as problems, this research then chooses the problem-centred initiation as its entry point since those challenges hinder learners in learning software modelling. Grounded on the identified problems, we could decide that there is a necessity to build a tool that can support software modelling students to develop their abstraction, diagramming, and algorithmic strategy skills in a more motivating and engaging way.

Define Objectives of a Solution. Based on the identified problems and motivations, we have defined our solution that is embedding gamification in the process of software modelling learning, which is expressed by the use of an artefact specifically designed with gamification and pedagogy in mind. The research objectives and outcomes are defined in section 3.2 and 3.3.

Design and Development. This research will employ gameful design method (Deterding, 2015) to design and develop the gamification and agile software development method to design and develop the artefact. The design and development activities are part of the iterative cycles and the products of the activities will be refined as required based the results generated from the evaluation activity.

Demonstration and Evaluation. The resulting artefact will be demonstrated and evaluated by applying it to several courses of software modelling. Moreover, the evaluation results will be used as feedbacks to improve the quality of artefacts and as a ground to judge the research findings. Demonstration and evaluation activities are parts of the iterative cycles and will be performed again as required.

Communication. Significant findings will be published in an academic conferences or journals for dissemination and evaluation by the related research communities.

2.4.2 Gameful Design Method

Deterding proposed a gameful design method that consists of gameful design steps, a set of design lenses, and skill atoms [27]. Gameful design method is a prescriptive guideline to implement gamification that comprises five phases: strategy, research, synthesis, ideation, and ideation.

In strategy phase, stakeholder define outcome metrics, target users, context, and activities as well as constraints and requirements. In the next phase, research phase, user activities behaviour chains, user needs, motivations, and hurdles as well as how game design fits with them are identified and defined. After that, in the synthesis phase, triplets or a combination of activity, challenges, and motivation is defined,including defining skill atoms. Furthermore, in the ideation phase, to generate ideas of the gamification design, brainstorming with innovation stems are conducted. Design lenses are used to help view the triplets from the perspective of game elements. After some ideas are generated, those ideas are prioritised. The prioritised ideas are then translated into storyboards to make them more visible. After that, the ideas are evaluated and refined based on results of the evaluation. In the last phase, a prototype is developed iteratively. This includes the activities of playtesting, analysing, building, and generating new ideas.

Design lenses is a combination of an easy-to-remember name, a short statement of a design principle, and a set of focusing questions that helps designer to view activities, motivation, and challenges from the perspective of the lenses. Design lenses are derived from game elements, so in short, viewing using design lenses means learning to view in the perspective of game elements. Deterding's Design

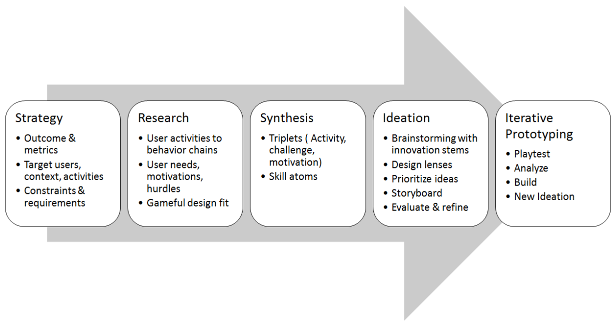
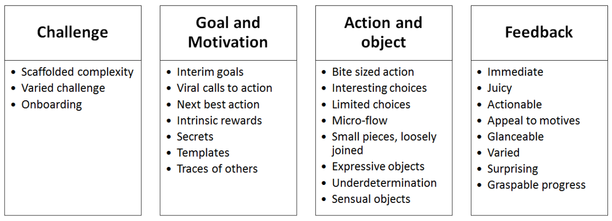

Strategy
• Outcome & metrics
• Target users, context, activities
• Constraints & requirements
Research
• User activities to behavior chains
• User needs, motivations, hurdles
• Gameful design fit
Synthesis
• Triplets (Activity, challenge, motivation)
• Skill atoms
Ideation
• Brainstorming with innovation stems
• Design lenses
• Prioritize ideas
• Storyboard
• Evaluate & refine
Iterative Prototyping
• Playtest
• Analyze
• Build
• New Ideation

Figure 2.13: Deterding's gameful design steps [27].

Lenses comes has four categories lenses: challenge, goal and motivation, action and object, and feedback, and each category comes with pre-existing lenses.



Challenge	Goal and Motivation	Action and object	Feedback
• Scaffolded complexity • Varied challenge • Onboarding	• Interim goals • Viral calls to action • Next best action • Intrinsic rewards • Secrets • Templates • Traces of others	• Bite sized action • Interesting choices • Limited choices • Micro-flow • Small pieces, loosely joined • Expressive objects • Underdetermination • Sensual objects	• Immediate • Juicy • Actionable • Appeal to motives • Glanceable • Varied • Surprising • Graspable progress

Figure 2.14: Deterding's design lenses [27].

A skill atom describes a feedback loop between user and system that is organized around a central challenge or skill: A user takes an action, which forms an input into the systems rule engine that determines state changes of its tokens, which get put out as feedback to the user, which she integrates into her mental model of the system. Through repeated interactionmultiple run-throughs of the atomthe user masters its skill: training handeye coordination, understanding the rules, and so on.

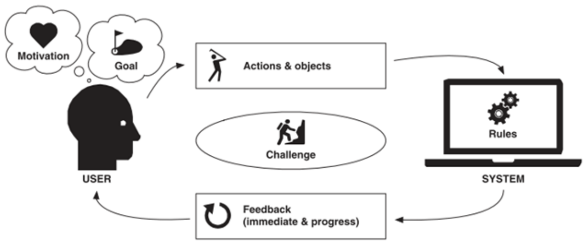


Figure 2.15: Deterding's skill atoms [27].

Chapter 3

Proposal

Grounded on the literature review in Chapter 2, we base our research in these premises. First, learning software modelling is not a trivial task as it requires abstraction skills and demands high cognitive loads to deal with abstract objects and operations. Second, regardless gamification design is still an ongoing challenge [28], it is an opportunity for research that up today there is still no work in the gamification of learning that addresses the core concepts of software modellingmodelling, metamodelling, and model transformation. Third, there is still no gamification design framework that guide the integration of game specific domain into software modelling learning. Fourth, opportunity to

3.1 Research Questions

The main research question proposed by this research is "How can gamification improve software modelling learning?" In order to support answering the main research question, following sub research questions need to be investigated:

1. Which processes, aspects, principles, or components of software modelling and their teaching and learning practices would benefit from gamification?
2. What types of game elements and in what roles can deliver software modelling learning best?
3. What kind of orchestrating framework is needed to design the interaction between software modelling and game elements to achieve software modelling gamification?
4. To what extent does gamification of software modelling improve learners' motivation, engagement, and performance?
5. To what extent do software modelling tutors benefit from software modelling game design framework?

3.2 Aim and Objectives

In order to answer the research questions, following aim and objectives should be met. The main aim of this research is to investigate and develop a software modelling game design framework that systematically and semi-automatically drives gamification design to produce software modelling learning games. More precisely, this research aims to meet the following research objectives that are derived from the main research aim:

1. Perform a literature review and surveys to identify research problems, questions, and objectives as well as necessary information to carry out the selected research methods and to develop an early version of the conceptual framework and artefact as the instantiation of the framework.
2. Develop a conceptual framework of how to design gamification of software modelling learning based on the literature review and the survey. The framework will be iteratively updated according to the results obtained from the experiments.
3. Develop a software artefact as the instantiation of the conceptual framework to produce a gamification of software modelling learning. The gamification will be tested to respondents for evaluation and to obtain feedbacks for iterative improvement.
4. Perform controlled experiments to measure the significance of the sofware modelling learning game in improving learning performance compared to the traditional method.
5. Perform controlled experiments to measure the productivity and maintenabilty of software modelling learning design framework in supporting tutors design and develop gamified software modelling learning.

3.3 Research Outputs

The potential research outputs of this research are:

1. Artefact. The software/application of gamification of software modelling learning.
2. Modelling Artefact. A tool for modelling the application of the gamification of software modelling learning.
3. Significance. Controlled experiments, learning outcome comparison between the gamified version and the traditional one.
4. Software Modelling Design Framework. Conceptual and software framework to perform gamification of software modelling learning.

5. Understanding. A model that explains how gamification of software modelling learning works. This could be achieved through Learning and Game Analytics and Structural Equation Modelling studies.

6. Case Study. Report the application of theories, models, and methods used in this research.

Chapter 4

Preliminary Results

4.1 Preliminary Survey

In this research, we have conducted our preliminary survey. The survey was executed to identify learners' needs, motivations, and hurdles according to the Research phase of the Deterding's gameful steps. This is also inline to the Design Science Research Methodology, which in the first activity we need to identify the problem and motivation so we can define objectives of a solution accurately in the second activity.

We have distributed online questionnaires to our respondents, students of Model Driven Engineering (MoDE) 2015/2016 module. The students were in their Software Engineering master programme at the University of York. From 21 students, only 4 completed the questionnaires. Their responses can be found in Appendix C.

Results. In order to identify the learners' needs, we asked our respondents question 1 and question 2. Question 1 aims to identify students' expectation before starting the module, while question 2 asked to identify what the students found important after taking the module. Based on the responses, the reasons why the students took MoDE module because they want to increase their knowledge on MDE, posses new advance skills or abilities, and improve their literation of MDE tools. After completing the module, the students valued that the most important lessons were getting new knowledge–domain modelling, metamodel, abstract syntax, abstract thinking, model validation, and the application of models–and skills–generating code, creating DSL, and improving their tool skills.

We also asked the students question 3, 4, and 5 to identify their motivation in taking MoDE module and learning Model Driven Engineering. Question 3 asked about the reasons behind their decision taking MoDE module. Two students stated that they took them module because it is compulsory, but the rest of the students said that they took the module because MDE is an advance topic and they wanted to see its applicability in industry and whether it will improve their ability–knowledge and skills.

Question 4 asked the students about what would motivate them more to learn Model Driven Engineering (MDE). The students gave feedback that they would be more motivated if they could perceive the advantages of MDE: efficiency and effectiveness it could give, the benefits of its application in the organization or real world examples, and its genericity–MDE application in languages other than Java or models other than UML/EMF.

We then asked question 5 which asked the students the most basic, underlying motives that make them commit to learn MDE. Substantially, they answered that their main motivation is to gain new ability–knowledge and skills, such as ability to make abstraction, advance skills that applicable in industry, and knowledge of reallife examples and applications of different models taught in MDE. Nevertheless, passing MoDE module, a pragmatical motives, still part of the whole motivation.

In question 6 and 7, we asked the students about the interesting challenges that they faced during MoDE module. They mentioned abstract thinking and model management activities such as defining metamodel, validating model, and how to best model a system–satisfying the model's metamodel and validation so the model could be easily queried and transformed). To overcome the challenges, what they did are performing trial-and-error method or experimenting the problems, try many other examples, and completing all the practicals. We summarised all of these efforts as activities to build 'experience'.

We also asked them question 8 and 9 about the non-interesting challenges, extraneous challenges that are not relevant to the core activities of modelling. They mentioned these activities: dealing with very specific technical concepts/words that only belong to specific products, focusing too much on how to use tools in other words using very tedious tools or less information on how to use the tools, and judging the quality of a model since there was no explanation of how good of bad the model was. To deal with the the uninteresting challenges, they just ignore them, seeking information and solutions from internet, lecturers, assistants, and discussion groups, and redid building the solutions from the beginning when they had certain problems using the tools.

Discussion. There are few takeaways from the preliminary interview in terms of students' needs, motivations, and challenges, that can be implemented into the design of a gamified software modelling learning. *First*, the need of the students to learn MDE is to gain new advance abilities–knowledge and skills–that are applicable in industry, which, if broken drown, they comprise of model management activities (abstract thinking, modelling, metamodelling, model transformation, validation, and application) and tool literacy.

Second, the the motivations of the students to learn MDE are, regardless their view on MoDE module as compulsary in their programme, they were aware that their motivation should be on satisfying their need in gaining advance knowledge and skills in MDE, as mentioned previously, and they would me more motivated if they could perceive the advantages and applicability of MDE, thus showing students the benefits and applications of MDE is very important in increasing their motivation.

Third, they mentioned model management activities (abstract thinking, model validation, metamodelling, etc.) as the interesting challenges. To overcome the challenges, they did trial-and-error method to gain more experience in overcoming the challenges. This is in line to experiential learning which states learning is at it's best through experiencing [7]. The main concern of gameful design is to reduce the cost of performing such activities so learners could focus on the core activities without hassles. This could be done by dividing the activities into smaller activity chunks and removing the extraneous, unrelated activities [27].

The extraneous, unrelated activities were identified by asking question 8 and 9, which was to identify the uninteresting challenges. Most of the complains were on the tools which were tedious to use. They also argued that there was no need to learn the detailed technical concepts or terms that were only specific to certain products. To overcome the uninteresting challenges, the students preferred to seek information and solutions from on internet, lecturers, and discussion groups. Thus, it is very important to provide comprehensive documentations and supports of the tools used in a learning activity [19].

4.2 Requirements

Category	Code	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, throughoutly, not deeply
	RL12	Teach with different modelling languages
	RL13	Make it fun
Tool Design	RL14	Support and documentation
	RL15	Build knowledge incrementally
	RL16	Flexibility to explore learning
	RL17	Positive reinforcement
	RL18	Convincing the value of learning
	RL19	High usability
	RL20	Inexpensive tool

4.3 Game Design

Gamification has been successfully for a variety of purposes, but there is very little work on software modelling gamification. We wish to assess whether gamification is beneficial for learners of graphical software modelling languages. For each modelling language, we envision the development of a dedicated game containing game elements that will be derived from the Lens of Intrinsic Skill Atoms [27]. The generated game will mimic a graphical modelling tool and at each level, it will require the learner to graphically construct or adapt a model to satisfy a set of requirements and constraints. 1 The game will have levels with 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 game to help learners familiarise themselves with the control system and the flow of the game.

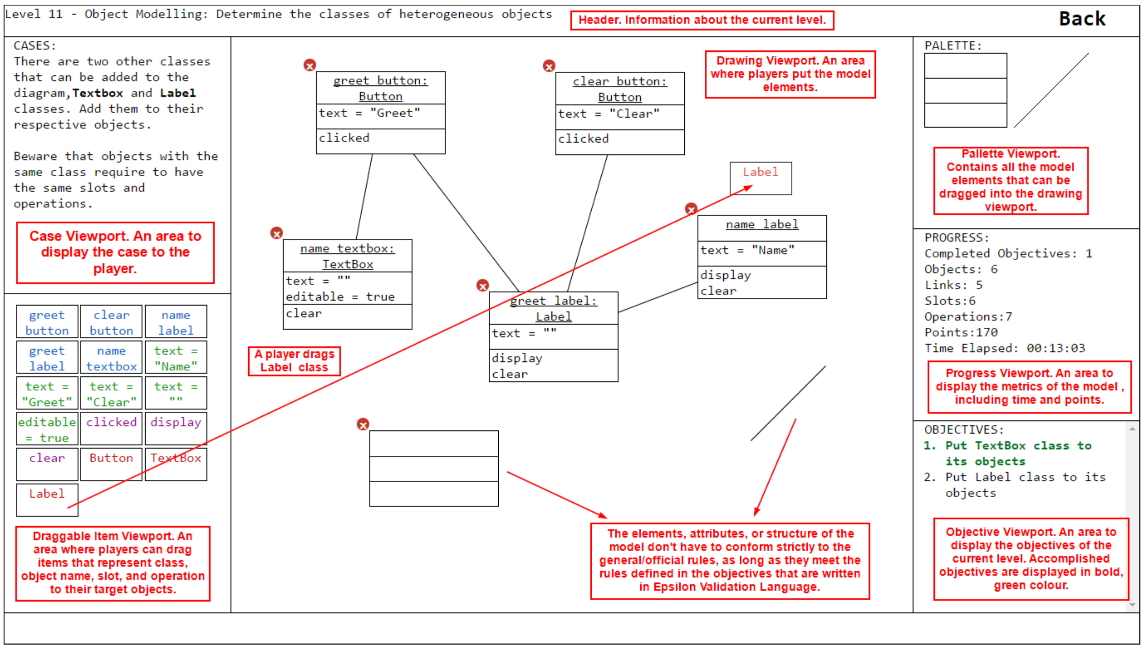
The game will include interim goals and intrinsic rewards to motivate learners. For software modelling, each type of modelling (e.g. object modelling, collaboration, process) will have several stories. A story will represent a specific case study to introduce learners to problems in specific domains. Every story will consist 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 a sense of step-by-step progress 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 very complex and time-consuming to model. Thus, the extraneous activities that are not relevant to the core concepts that are being taught should be removed. As a result, learners will be more focused on the main concepts. Thus, game elements like bite-sized actions (e.g. drag and drop), limited choices (i.e. only limited items can be dragged), and microflows (i.e. put the right element to its right place) will be implemented to facilitate learners in performing the core activities. Likewise, fuzziness will also be used to provoke 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 game. Games should be able to give immediate, glanceable, and actionable feedback to keep learners on track and monitor their progress. Interesting and varied feedback should be designed to appeal to the learners' motives.

To reduce bias, we plan to experiment with several modelling languages (e.g. BPMN, state-charts, GSN, UML). We also plan to implement these games using web technologies so that they are easily accessible to a wide audience.

4.4 Modelling Game Design Framework

Instead of developing the software modelling games manually, we plan to follow a model-based approach. We will use metamodel annotations, in the spirit of Eugenia [1], to define the graphical syntaxes of modelling languages and separate models to specify the game elements (levels, objectives, constraints, etc.) of each game. These models will be then consumed by a model-to-text transformation to produce fully-functional language-specific games. Therefore, the framework supports software modelling tutors in the design and customisation of the games at the high level of abstraction and so as to automatically build the game. So far we have implemented a metamodel for specifying game elements (flows, levels, challenges, and objectives) and a supporting Eclipse-based graphical editor (Fig. 4.2), and a prototype game (Fig. 4.1) for object diagrams.


Level 11 - Object Modelling: Determine the classes of heterogeneous objects

Header. Information about the current level.

Back

CASES:
There are two other classes that can be added to the diagram, **Textbox** and **Label** classes. Add them to their respective objects.

Beware that objects with the same class require to have the same slots and operations.

Case Viewport. An area to display the case to the player.

Draggable Item Viewport. An area where players can drag items that represent class, object name, slot, and operation to their target objects.

A player drags Label class

Drawing Viewport. An area where players put the model elements.

The elements, attributes, or structure of the model don't have to conform strictly to the general/official rules, as long as they meet the rules defined in the objectives that are written in Epsilon Validation Language.

PALETTE:

Pallette Viewport. Contains all the model elements that can be dragged into the drawing viewport.

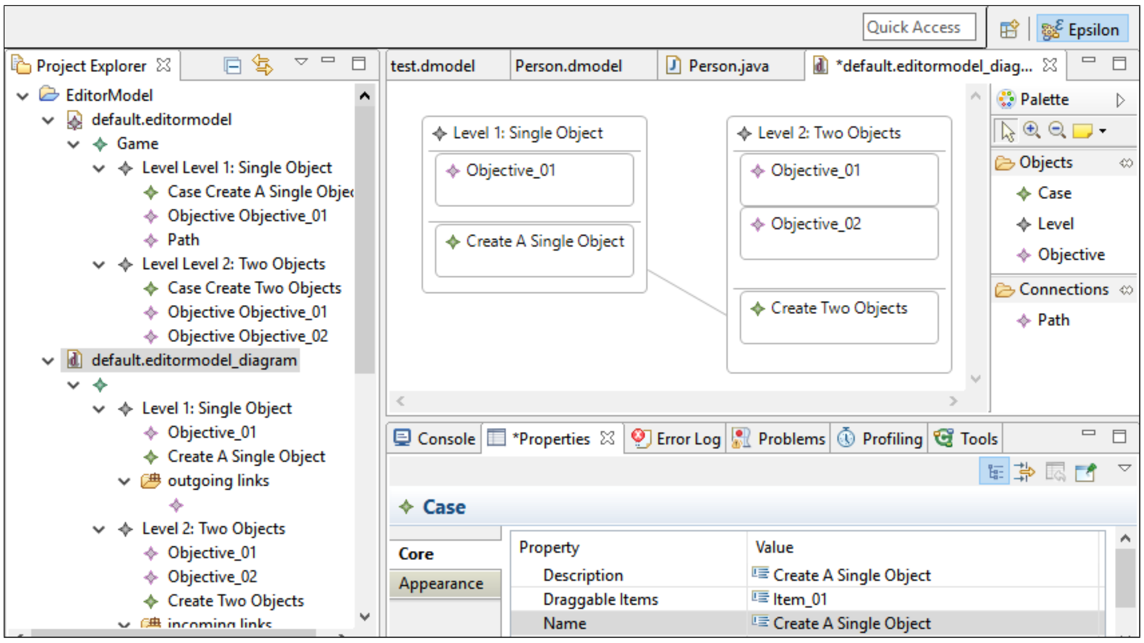
PROGRESS:
Completed Objectives: 1
Objects: 6
Links: 5
Slots:6
Operations:7
Points:170
Time Elapsed: 00:13:03

Progress Viewport. An area to display the metrics of the model, including time and points.

OBJECTIVES:
1. **Put TextBox class to its objects**
2. Put Label class to its objects

Objective Viewport. An area to display the objectives of the current level. Accomplished objectives are displayed in bold, green colour.

Figure 4.1: The display of the generated game.



Property	Value
Description	Create A Single Object
Draggable Items	Item_01
Name	Create A Single Object

Figure 4.2: Graphical editor for the game specification DSL.

Bibliography

[1] 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.

[2] M. Csikszentmihalyi, "Toward a psychology of optimal experience," in *Flow and the foundations of positive psychology.* Springer, 2014, pp. 209–226.

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

[4] 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.

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

[6] C. Munzenmaier and N. Rubin, "Blooms taxonomy: Whats old is new again," *The Elearning Guild. Santa Rosa*, 2013.

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

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

[9] R. Hunicke, M. LeBlanc, and R. Zubek, "Mda: A formal approach to game design and game research," in *Proceedings of the AAAI Workshop on Challenges in Game AI*, vol. 4, 2004, p. 1.

[10] K. Werbach and D. Hunter, *For the win: How game thinking can revolutionize your business.* Wharton Digital Press, 2012.

[11] 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.

[12] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis Lectures on Software Engineering*, vol. 1, no. 1, pp. 1–182, 2012.

[13] 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.

[14] 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.

[15] I. Ober, “Teaching mda: From pyramids to sand clocks,” in *Symposium at MODELS 2007*. Citeseer, 2007, p. 34.

[16] 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.

[17] 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.

[18] 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.

[19] 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,” 2015.

[20] T. C. Lethbridge, “Teaching modeling using umple: Principles for the development of an effective tool,” in *2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 2014, pp. 23–28.

[21] 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.

[22] 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.

[23] 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.

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

[25] O. Richardsen, “Learning modeling languages using strategies from gaming,” Master’s thesis, Norwegian University of Science and Technology, Norway, 2014.

[26] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[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] S. Deterding, S. L. Björk, L. E. Nacke, D. Dixon, and E. Lawley, “Designing gamification: creating gameful and playful experiences,” in *CHI’13 Extended Abstracts on Human Factors in Computing Systems.* ACM, 2013, pp. 3263–3266.

[29] A. Yohannis, “Gamification of software modelling,” in *the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016) Doctoral Symposium.* CEUR, 2016.

Appendix A

Research Plan

The planned research schedule can be found in Table 1. In three years, this work plans to have five times iteration of Design-Develop-Demonstration-Evaluation, with two of them will be carry out in Indonesia and the rest of them in York.

Table A.1: Research Timetable

Month	2016	2017	2018
1	Literature Review	**Progress Report**	(in Indonesia or online) - 4th Application & Evaluation - 5rd Survey
2		(in Indonesia or online) - 2nd Application & Evaluation - 3rd Survey	
3			
4	- **25-minute seminar** - Design Questionnaires - 1st Survey		
5			
6	- Develop Framework - Develop Prototype - **Qualifying Dissertation**		- **First Thesis Audit** - Update Framework - Update Prototype - Update Questionnaires
7		- Update Framework - Update Prototype - Update Questionnaires - **Thesis Outline**	
8			
9	- 1st Application & Evaluation - 2nd Survey	- 3rd Application & Evaluation - 4th Survey	- 5th Application & Evaluation - 6th Survey
10			
11			
12	- Update Framework - Update Prototype - Update Questionnaires	- Update Framework - Update Prototype - Update Questionnaires	- **40-minute Seminar** - **Thesis**

Appendix B

Publications

This research has been published in the following conferences or journals:

1. A. Yohannis, Gamication of software modelling, in the ACM/IEEE 19th International Con- ference on Model Driven Engineering Languages and Systems (MODELS 2016) Doctoral Sym- posium. CEUR, 2016 [29].

Appendix C

Preliminary Survey Data

1. **If you think back to the time when you were just about to start the MODE module, what did you think you would find interesting in Model Driven Engineering?**

    - Learning what MDE actually is. Had never heard of it so was intrigued.
    - I thought I would be provided with a high level approach to designing and managing software/critical systems.
    - Learning some tools to create automodels.
    - Learning about ways of statically verifying that code conformed to a formal model, and using this to detect and/or automatically correct bugs learning about ways to automatically modify code based on changes made to a model.

2. **What did you find important in Model Driven Engineering?**

    - Domain modelling especially metamodels and the whole concept of abstract syntax. In terms of practicalities, the ability to use models to generate code is the most useful, along with creating DSLs.
    - Trying to learn the specific tools to pass the assessment. Not ideal as I wanted more generic skillsets in this domain I could apply in my career, instead it was too focused on learning some niche features in Epsilon.
    - abstract thinking validate the models linking the model to reallife.
    - The ability to keep a formal model that the code has been verified to conform to throughout an entire development process Automatic code generationmodifcation based on a model.

3. **Why did you decide to take the Model Driven Engineering module?**

    - It was compulsory. I had no choice.
    - Compulsory.
    - well, I feel that everything around me has a certain model, therefore, i felt learning about. Model Driven will increase my knowledge and experience in work.

- It seemed like it would be useful to learn a new approach to software engineering and skills that might be valuable in industry in the future I was interested in generating and transforming code automatically and using a formal model for the structure of code.

4. **What would motivate you more to learn Model Driven Engineering?**

    - Seeing the MDE approach being used to do something that would otherwise be much more tedious to do using more conventional means.
    - To see the benfit applied in the real world and how organisations have benefited from it. Then how can use these skills and adapt them to my needs.
    - If we link it to reallife examples, explore more other tools.
    - More use of it in industry More use with languages other than java and different types of models such as ones that aren't based on UML/EMF.

5. **Based on the answers that youve provided above (No. 14), what were the most basic, core underlying motives or needs that make you commit to learning Model Driven Engineering?**

    - The ability to think at a higher level of abstraction and understand the concepts which link together a domain especially, for example, programming languages.
    - Passing the compulsory paper.
    - see some reallife examples and apply different models to see the differences.
    - Learning skills which would be valuable in the SE industry in the future Learning something which would help with controlling the complexity of a SE process by making sure that code conforms to a formallydefined model

6. **What were the challenges that you found interesting in learning ModelDriven Engineering? Why?**

    - One of the main challenges is in defining the abstract syntax for a DSL along with placing restrictions on its use through a validation language. There's a balance between trying to make the abstract syntax clean and easy to understand and modify vs. preserving the intended semantics.
    - None really, I found the assignment and practicals a tool based grind as opposed to a useful learning opportunity.
    - I felt sometimes that I could not apply all principles in the practicals especially on how we think abstract.
    - Working out how to best model a system, which had been defined informally, in EMF and EVL while keeping all its constraints intact and making it easily queryable and transformable was interesting.

7. **How did you manage to overcome these challenges?**

   - By experimenting and going with what makes the most sense. If it's a structural issue with semantics, it's an abstract syntax issue. If it's something more peculiar, it's a validation issue.
   - By grinding through them.
   - I tried to train myself on other examples, but still I could not link my models to reallife example(as real project).
   - The best way to learn this was from experience which was gained by completing all the practicals.

8. **What were the hurdles–the noninteresting challenges–that hindered or demotivated you in learning ModelDriven Engineering? Why?**

   - Learning the Eclipse Modelling Framework, MOF etc. wasn't fun. The most fun part was learning and use Emfatic with EGL/EGX and EOL in general. But actually understanding the metametamodel and all the ecore stuff seemed pointless. You don't need to understand what EClass, EEnum, EString etc. are. It's just unnecessary detail that's very specific to Eclipse and not really something that you need to know even if you use Epsilon.
   - The focus on the tool as opposed to the high level concepts and skill sets that would empower me to utilise model driven engineering in the real world.
   - sometime the tool itself, you need to retrack all your changes manually, no right answer or a good explanation why this model is good or bad.
   - Problems using eclipse, the shortcomings of EMF, lack of information available on the internet Eclipse is very large, complex and fragile. EMF/ UML style models can be quite restrictive at times when modelling complex relationships, and often requires resorting to EVL. This is annoying because EVL constraints cannot be easily displayed on a diagram and two different languages/systems, EVL and EMF, are being used for similar things. Sometimes two very similar constraints exist where one can be modeled in EMF and the other can't and so requires EVL. Although there is a lot of very good documentation available about the languages in epsilon there is far less information on the internet about them than what is usually available for popular programming languages and it qould be helpful if there was more.

9. **How did you overcome these hurdles?**

   - By ignoring them once I realized they served no purpose for developers.
   - Reading up about the tools and grinding through them.
   - Asking questions, reading some examples in the Epsilon website forum.

- Eclipse sometimes stops working but does this usually does not prevent the completion of tasks as most problems can be fixed by deleting the workspace directory and starting again, it simply wastes a lot of time Things that could not be expressed in EMF were instead expressed using EVL. The required information about epsilon could always be found by asking classmates and asking lecturers, but this would not be possible if using these languages outside the university