**Progress Report**

# Model-Driven 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

March 9, 2017

**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. We plan to perform controlled experiments to evaluate the effectiveness of the platform as well as the gamified software modelling learning produced.

# Contents

**Abstract** 1

**Contents** 2

**1 Introduction** 3
- 1.1 Background . . . 3
- 1.2 Research Questions . . . 5
- 1.3 Research Objectives . . . 6
- 1.4 Research Outputs . . . 6
- 1.5 Research Scoping . . . 7

**2 Progress Review** 8
- 2.1 Research Methods . . . 8
- 2.2 Identify Problems and Motivation . . . 9
  - 2.2.1 Problems and Motivation . . . 9
- 2.3 Define Objective for a Solution . . . 9
  - 2.3.1 Requirements from a Survey . . . 10
  - 2.3.2 Requirements from Literature Review . . . 12
- 2.4 Design and Development . . . 13
  - 2.4.1 Design . . . 13
  - 2.4.2 Development . . . 18
- 2.5 Demonstration and Evaluation . . . 18
- 2.6 Communication . . . 18

**3 Research Plan** 19

**4 References** 20

**5 Publications** 23

**Appendix A Preliminary Survey Data** 24

**Appendix B Application of Deterding's Gameful Design Steps** 28

B.1 Strategy . . . 28
B.1.1 Define Target Outcome and Metrics . . . 28
B.1.2 Define Target Users, Context, Activities . . . 28
B.1.3 Identify Constraints and Requirements . . . 29
B.2 Research . . . 29
B.2.1 Translate Users Activities into Behaviour Chains . . . 29
B.2.2 Identify User Needs, Motivations, challenges . . . 30
B.2.3 Determine Gameful Design Fit . . . 31
B.3 Synthesis . . . 31
B.3.1 Formulate Activity, Challenge, Motivation Triplets for Opportune Activities or Behaviours . . . 31
B.4 Ideation . . . 32
B.4.1 Brainstorming Ideas using Innovation Stems . . . 32
B.4.2 Prioritise Ideas . . . 33
B.5 Iterative Prototyping . . . 33

**Appendix C Storyboard** **34**

# Chapter 1

# Introduction

In this section, the background, questions, objectives, potential outputs, and scoping of this research are presented.

## 1.1 Background

Software modelling is commonly perceived as a difficult subject since it requires a mastery of abstraction [1]. However, this subject has a significant role in software engineering education and practice. Successful application of software modelling requires skills in abstract modelling [2]. The modelling itself is the process of thinking abstractly about systems [3]. Thus, teaching modelling also means teaching abstraction [4]. Therefore, it is crucial to make students understand the value of abstraction [3]. 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 [5]. Drawing from multidiciplinary research to define abstraction for Artifical Intelligence, Saitta et al. stated that abstraction is often associated with these processes: information hiding, generalisation, approximation or reformulation, and separating relevant from irrelevant aspects [6]. Those processes are the common approaches applied in Computer Science, such as encapsulation and generalization in object-oriented programming, resolution of polygon density in 3D modelling, algebraic simplication in symbolic computation, and removing colour information of images to produce grayscales. In the context of software engineering and computer science education, Kramer [5] and Hazzan [7] argued that abstraction is the central theme or key skill for computing.

> "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 ... — Kramer [5].

> " ... software is an intangible object, and hence, requires highly developed cognitive skills for coping with different levels of abstraction." — Hazzan [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]. The abstract nature of software modelling traditionally has been addressed through the use of diagrams or visual notations in the form of modelling and diagramming tools. However, grasping concepts represented by the visual notations and conversely presenting back the concepts into visual notations are not trivial, including presenting aspects of the concept in different visual notation as well as choosing the right levels of abstraction and moving between them. Learning and teaching these skills is challenging and to overcome the challenges, we need to put effort into software modelling learning platform that support learning the skills.

The presence of learning platform does not guarantee will improve learners' engagement in software modelling learning. In recent years, the use of games and game elements, such as Gamification [9] and Serious Games [10], for purposes other than leisure has drawn significant attention. Gamified approaches 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. Figure 1.1 shows the growth of "Gamification" on Google Trends[1] since February 2010.

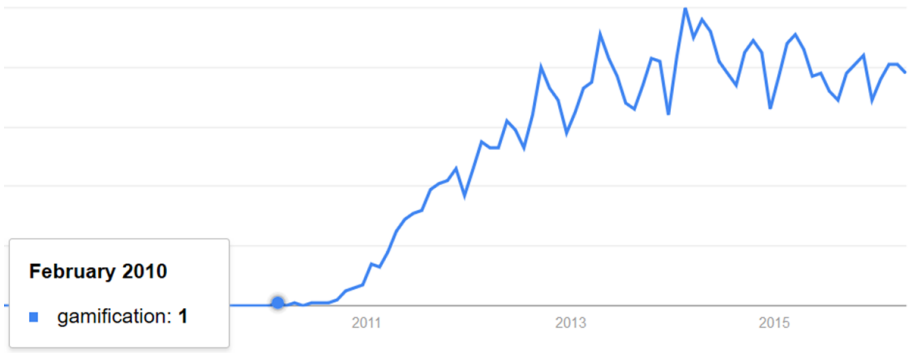


Figure 1.1: "Gamification" on Google Trends per 12 April 2016.

Real-world examples that show the success of the gamification are Duolingo[2] and Re-mission[3]. 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. [11] 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. [12] found that according to the majority of the reviewed papers,

[1] https://www.google.com/trends/explore?date=all&q=%22gamification%22
[2] https://www.duolingo.com/
[3] http://www.re-mission.net/

gamification does generate benefits and positive effects. Specifically in the field of software engineering, Pedreira et al. [13] 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 gamification in software engineering. Reports of the positive impact of gamification in various fields, particularly addressing engagement issues, encourage us to leverage it to deliver gamified software modelling learning, an area which has received little attention for the application of games and game elements so far.

In terms of learning, pedagogical aspect cannot be neglected. Several concepts from pedagogy will be applied to drive the design of the platform, particularly the best practices from instructional design, a mature field in designing learning activities so that learning processes can be more efficient and effective. Therefore, incorporating gamified approaches as well as the best practices of instructional design will bring great benefits to tackle the problems in learning software modelling. The platform is being designed to be suitable for higher-level undergraduate and postgraduate students with some of experience of software engineering.

Rather than just being a learning content for learners, software modelling also broadens opportunity to improve the meta-processes of the gamified software modelling learning, through the application of Model-Driven Engineering (MDE) approaches. Instead of developing the software modelling games manually, a model-based approach is applied. A platform is being developed and it will act as a design framework to design learning activities for topics in software modelling. The learning activity design then will be transformed to generate gamified learning activities of software modelling.

## 1.2 Research Questions

Thus, this research proposes a main research question "What kind of platform that produce gamified learning activities to improve software modelling learning?". The word 'platform' means a software environment for tutors and learners to design and perform gamified software modelling learning. The word 'produce' indicates the platform support tutors in designing and generating gamified learning activities. Thus, the platform should be expressive enough to support various visual modelling notations and the creation of different patterns of learning activities for learning software modelling. The word 'improve' implies that learning with gamified approaches enhances learners' engagement and learning performance. They are 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. To answer the main research question, following sub research questions need to be investigated:

1. How can a platform that implements Model-Driven Engineering approaches help tutors and learners to produce and learn gamified software modelling learning?

2. Does the platform improve learners' engagement and performance in learning software modelling?

3. Does the platform help tutors in developing gamified software modelling learning?

## 1.3 Research Objectives

The solution proposed by this research is to produce a platform that can support tutors to design gamified learning activities as well as learners to learn software modelling in a gamified way. More precisely, this research aims to meet the following research objectives that are derived from the solution:

1. Design and develop a platform that accommodates gamified approaches and instructional design to be implemented through harnessing the best practices of Model-Driven Engineering.

2. Perform controlled experiments to measure the significance of the platform in improving learning engagement and performance compared to traditional method, didactic learning without the support of gamified approaches.

3. Perform controlled experiments to measure the productivity and maintainability of the platform regarding supporting tutors in designing and producing gamified learning activities of software modelling learning.

## 1.4 Research Outputs

By the end of this research, three potential research outputs have identified will have been produced:

1. A platform for tutors to design and produce gamified learning activities of software modelling and for learners to learn software modelling in gamified ways.

2. Controlled experiments: comparisons in learning engagement and performance of learners between gamified version and the traditional one of software modelling learning.

3. Controlled experiments: comparisons in productivity and maintainability of of the platform in supporting tutors designing and producing gamified software modelling learning activities.

## 1.5 Research Scoping

In the beginning, this research plans to address software modelling in Model-Driven Engineering as a whole—comprising modelling, metamodelling, and model transformation. However, after consideration regarding scope and time, it's scope is adjusted to focus on graphical software modelling, which is a common way to express models in modelling and metamodelling. Model transformation is excluded since its approaches are commonly expressed in a textual way, but still includes metamodelling since a metamodel itself is a model of models and usually is expressed in the form of class diagram-like graphics. This research plans to perform literature study and develop a prototype in the first and a half year and address experiments for validation in the second year and third year respectively.

The remainder of this report is organised as follows. We provide a detailed progress review in Section 2 and a research plan in Section 3. Section 4 contains the references of this report and finally Section 5 lists the publication of this research.

# Chapter 2

# Progress Review

In this section, the progress review of this research is discussed. First, the Design Science Research Methodology (DSRM) [14], the main research method employed on this research, is discussed briefly, and then followed by a discussion of progress on activities composing DSRM.

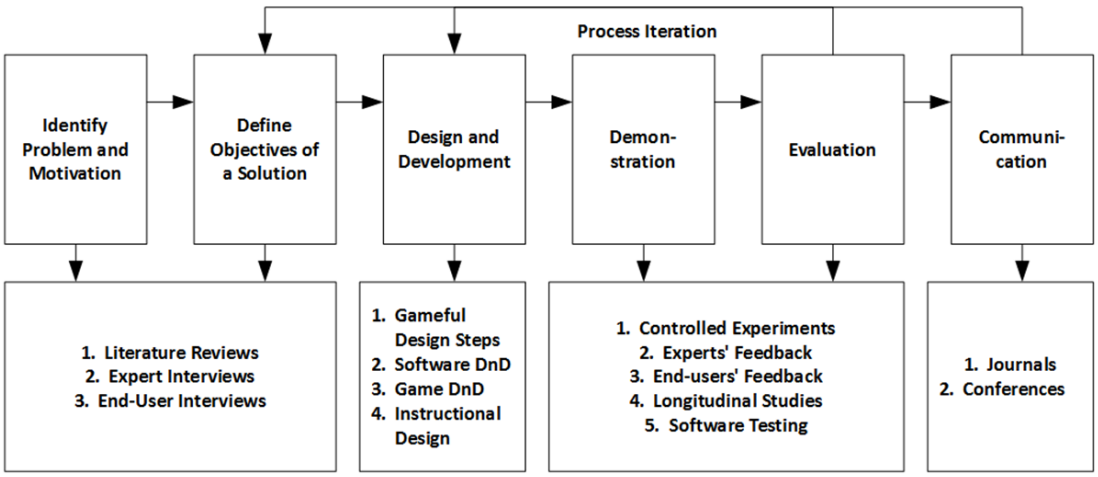


Figure 2.1: Design Science Research Methodology. Adapted from Peffer et al. [14].

## 2.1 Research Methods

Design Science Research Methodology (DSRM) is selected as the main research method since the main output of this research is a design artefact, a platform to design and generate gamified software modelling learning. DSRM provides a comprehensive conceptual framework and activity guidelines for understanding, developing, executing, and evaluating a design artefact (Figure 2.1). Another reason is that it positions itself at the top level of abstraction without going into much detail of how to perform each activity, 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. DSRM consists of six activities: identify problem and motivation, define objectives for a solution, design and development, demonstration and evaluation, and communication. The progress of this research on these six activities are discussed in the following sections.

## 2.2 Identify Problems and Motivation

Problem identification and motivation of this research have been largely presented in the Introduction section (Chapter 1). In this section, they are restated again in a succinct manner.

### 2.2.1 Problems and Motivation

Software modelling plays a significant role in software engineering. However, its abstract nature make it challenging to be mastered since most of its concepts are largely presented in symbolical notations. To master software modelling, learners need to posses modelling skills, such as grasping ideas behind models presented in visual modelling notations, presenting back the ideas into visual modelling notations, viewing the ideas from different perspectives using different visual modelling notations, choosing the right levels of abstraction of modelling and moving between. Acquiring these skills might be difficult enough to demotivate learners from learning software modelling.

Apart from the problems previously mentioned, the use of games or game elements rises as solutions to tackle motivational problems. Also, instructional design has been a mature field in designing learning activities so that learning processes can be more efficient and effective. Therefore, incorporating gamified approaches as well as the best practices of instructional design will bring great benefits to tackle the problems in learning software modelling. A platform that unify the gamified approaches and instructional design is needed. This is where software modelling plays it role. Not just being a content to be learned, using Model-Driven Engineering best practices, the design of gamified learning activities could be automated, customised, and reused, and the gamified learning activities themselves could be more engaging compared to the traditional didactic learning. Thus, this research is motivated to produce a platform that can support tutors to design gamified learning activities as well as learners to learn software modelling in a gamified way.

## 2.3 Define Objective for a Solution

The objective of the solution proposed by this research is to develop a platform that can support tutors to design gamified learning activities as well as learners to learn software modelling in a gamified way. To ensure the solution will be fully

effective, some requirements have to me met. These requirements are gathered from two sources, from a preliminary survey and literature review.

### 2.3.1 Requirements from a Survey

In this research, we have conducted our preliminary survey to identify learners' needs, motivations, and challenges according to the Research phase of the Deterding's Gameful Design Framework. The preliminary survey is not intended to measure significance, but it is more to qualitative investigation to reveal needs, motivations, and challenges in learning software modelling from learners' perspective. The preliminary survey is also in line with the Design Science Research Methodology, in order to identify the problem and motivation so that we can define objectives of a solution accurately in the second activity.

We have distributed online questionnaires to 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 A. Since the number of respondents are small for generalisation, we plan to apply the same survey to next term MODE students.

**Results**. To identify the learners' needs, we asked our respondents two questions. Question 1 aims to identify students' expectations before starting the module, while question 2 aimed at identifying 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, possess new advanced 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 three questions (Q3-Q5) 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 the module because it is compulsory, but the rest of the students said that they took the module because MDE is an advanced topic and they wanted to see its applicability in the 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 responded that they would be more motivated if they could perceive the advantages of MDE: efficiency and effectiveness it could offer, the benefits of its application in the organisation 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 learning MDE. Substantially, they answered that their main motivation is to gain new ability—knowledge and skills, such as the

ability to make an abstraction, advanced skills that are applicable in industry, and knowledge of real-life examples and applications of different models taught in MDE. Nevertheless, passing MODE module, a pragmatical motive, 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 metamodels, 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 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 following 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 uninteresting challenges, they just ignored them, seeking information and solutions from the 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 findings from the preliminary interview regarding students' needs, motivations, and challenges, that can be implemented into the design of a SMLG. *First*, the need of the students to learn MDE is to gain new advanced 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 motivations of the students to learn MDE are, regardless their view on MODE module as a compulsory module in their programme, they were aware that their motivation should be on satisfying their need in gaining advanced knowledge and skills in MDE as mentioned before, and they would be more motivated if they could perceive the advantages and applicability of MDE, thus showing students the benefits and applications of MDE are crucial 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. These findings are in line with experiential learning which states learning is best achieved through experiencing [15]. The main concern of gameful design is to reduce the cost of performing such activities so that learners could focus on the core activities without distraction. It could be done by dividing the activities into smaller activity chunks and removing the extraneous, unrelated activities [16].

The extraneous, unrelated activities were identified by asking question 8 and 9, which aimed at determining the uninteresting challenges. Most of the complaints

Table 2.1: Requirements derived from the preliminary survey (section **??**).

| Category | Code | Requirements from Preliminary Survey |
|---|---|---|
| Needs | RS01 | Teach them knowledge and skills that are applicable in industry: model management (abstract thinking, modelling, metamodelling, model transformation, validation, and application) and tool literacy. |
| Motivations | RS02 | Promote gaining advance knowledge and skills in MDE. |
| | RS03 | Promote the benefits and applications of MDE. |
| Interesting Challenges | RS04 | Challenge with model management activities (abstract thinking, model validation, metamodelling, etc.). |
| | RS05 | Scaffold learning process to support learners gaining their experience (for an example, dividing the activities into smaller activity chunks). |
| Uninteresting Challenges | RS06 | Increase the usability of the tool being used. |
| | RS07 | No need to learn the detailed technical terms specific to certain products. |
| | RS08 | Provide documentation and support for the tools being used. |

were on the tools which were tedious to use. They also argued that there is no need to learn the detailed technical concepts or terms that were only unique to certain products. To overcome the uninteresting challenges, the students preferred to seek information and solutions from on internet, lectures, and discussion groups. Thus, it is paramount to provide comprehensive documentation and support of the tools used in a learning activity [17].

### 2.3.2 Requirements from Literature Review

Throughout the literature review, we have gathered requirements, which are the key points pointed out by MDE experts how we should teach MDE. Table 2.2 is a list of requirements summarised from the literature review in section **??**. These requirements have two roles. First, they provide guidance to our design process and, second, they will also act as units of evaluation to confirm the SMLG meets the current best teaching practices.

We also derive other requirements from our preliminary survey in section **??**. These requirements (Table 2.1) have the same role as other requirements derived from the literature review, except that these requirements address the needs, motivations, and challenges in designing the gameful aspect of SMLG. From our requirement identification, we found out that the items in the Contents category (Table 2.2) agrees with the item in the Needs category (Table 2.1). The finding suggests an agreement about the learning contents that should be delivered to students in learning MDE.

Table 2.2: Requirements derived from the literature review (section **??**).

| 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 |

## 2.4 Design and Development

### 2.4.1 Design

#### Pedagogical Aspect Design

In section **??**, we proposed several existing learning models that we will apply in the design process of our SMLG. 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 SMLG are depicted in in Figure 2.2 and Figure 2.3.

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 SMLG. 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 2.2). Bloom's taxonomy also act as an inventory of activities that provide us many options of activities that could give variability in our design.

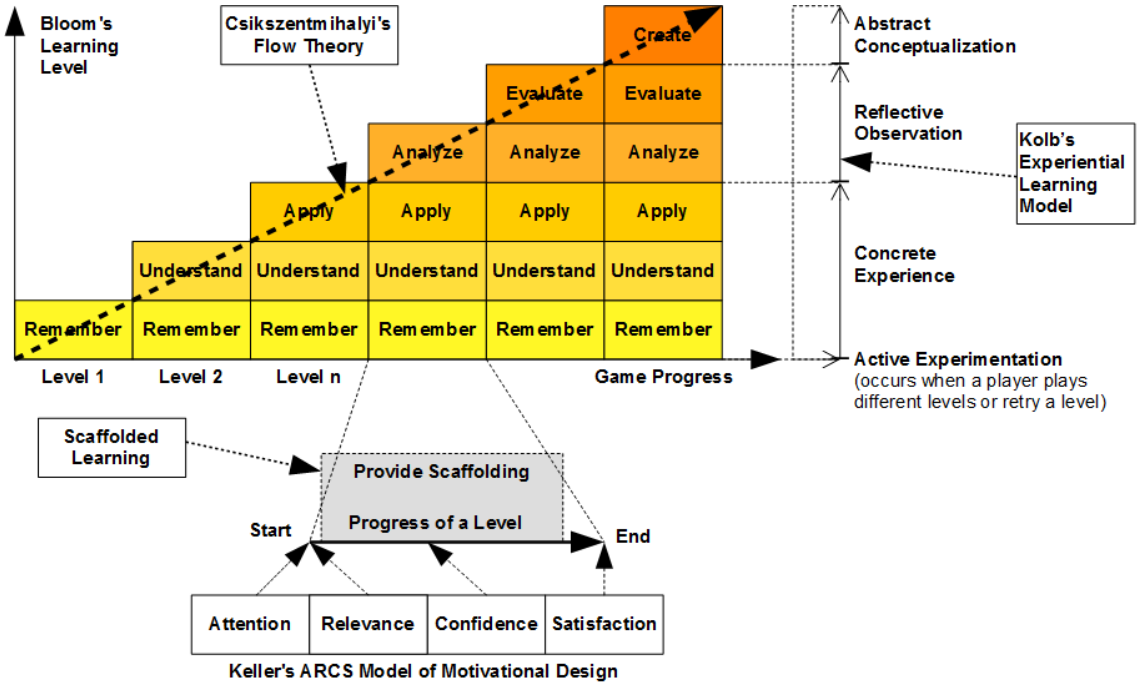


Figure 2.2: Elaborating learning models' contribution to the design of the gamified modeling learning

.

While learners are progressing in the SMLG, 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 2.2). 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 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 2.2. 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 [18]. 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 SMLG, there are a start, an end, and learning activities in between (Figure 2.2). We could apply the

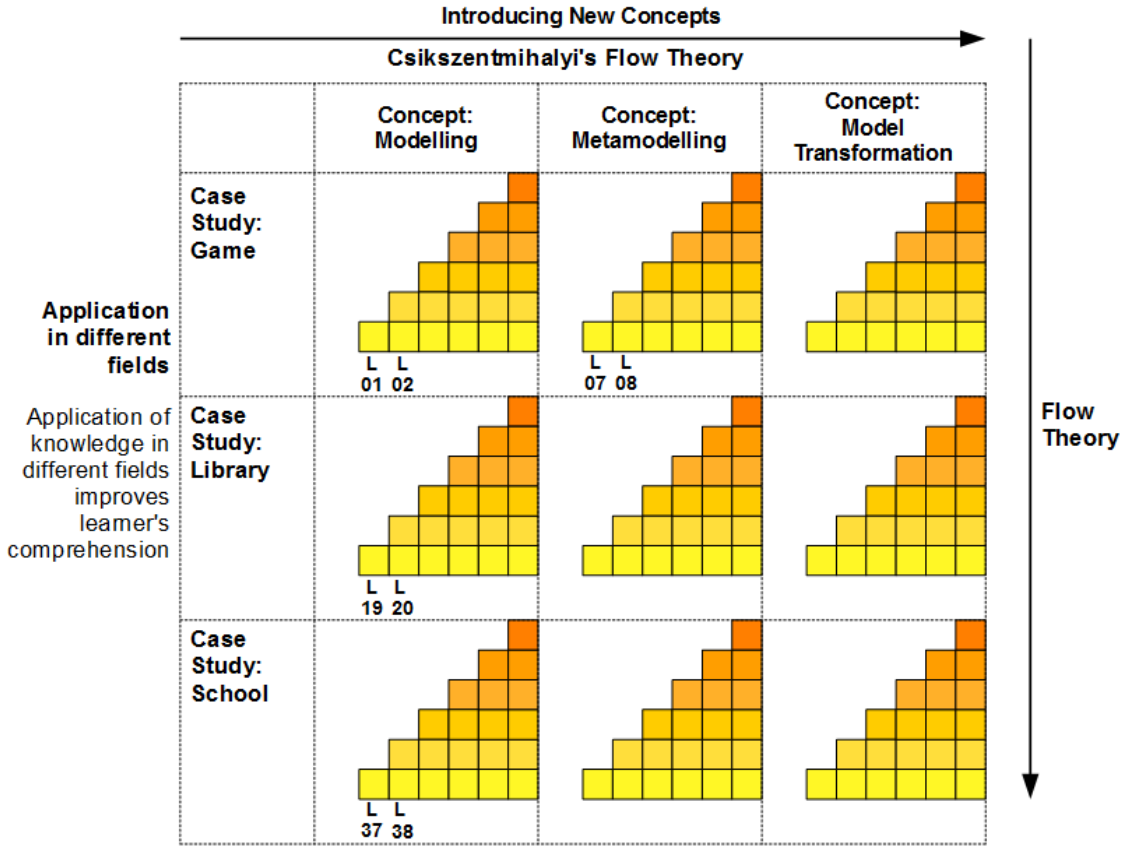


Figure 2.3: Elaborating learning models' contribution to the design of the gamified modeling learning

.

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 [19, 20] to support learners coping challenges (Figure 2.2). 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 [15]. We select Kolb's model since we perceive that playing a level in SMLG 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).

We could apply this cycle 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). In case that the player already 'game Over' and they cannot apply their new solution, AE occurs when they replay the same level or play a similar level.

Table 2.3: 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 [21] and both have been applied together to design instructions in different fields [22, 23, 24]. 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 2.3) that will be used in the design and evaluation activities.

### 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[1] and BPMN[2], 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 SMLG, we plan to experiment and support several graphical modelling languages (e.g. UML, BPMN, state-charts). For each modelling

[1] http://www.uml.org/
[2] http://www.bpmn.org/

language, we envision the development of dedicated SMLG that will be derived from the Gameful Design Framework [16]. The SMLG 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 SMLG 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 SMLG to help learners familiarise themselves with the control system and the flow of the SMLG.

The SMLG 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 SMLG. The SMLG 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 SMLG 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 B. 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 framework that will facilitate the design and generation of SMLG. Rather than developing SMLG for each graphical modelling language manually, we will follow a model-based approach. In the spirit of Eugenia [25], 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 SMLG. These models will be then consumed by a model-to-text transformation to produce fully-functional language-specific SMLG. Thus, the framework supports

software modelling tutors in the design and customisation of the SMLG at the high level of abstraction as well as to automatically build the SMLG. Up to now we have implemented a metamodel to specify game elements (flows, levels, challenges, and objectives) and a supporting Eclipse-based graphical editor (Fig. **??**), and a prototype SMLG (Fig. **??**) for object diagrams.

### 2.4.2 Development

## 2.5 Demonstration and Evaluation

## 2.6 Communication

The progress of this research should be communicated to a larger audience. So far, only one publication have been made in a doctoral symposium setting. The detail of the publication can be found in the Publications section (Chapter 5).

# Chapter 3

# Research Plan

We plan to complete literature study and develop prototype by the end of the first year (2016), and address gamification of modelling and metamodelling in the second year (2017) and third year (2018) respectively.

Table 3.1: Research Timetable

<table>
<tr><th>Month</th><th>2016</th><th>2017</th><th>2018</th><th>2019</th></tr>
<tr><td>01</td><td rowspan="3">Literature Review</td><td rowspan="3">- Develop Prototype<br>- 1st experiment<br>- 2nd survey</td><td rowspan="3">- Update Prototype<br>- 3rd experiment<br>- 4th survey</td><td rowspan="11">Thesis writing</td></tr>
<tr><td>02</td></tr>
<tr><td>03</td></tr>
<tr><td>04</td><td rowspan="2">- 25-minute Seminar<br>- Questionnaire Design<br>- 1st survey</td><td>Progress Report</td><td>Thesis Audit</td></tr>
<tr><td>05</td><td rowspan="4">(In Indonesia)<br>- Update Prototype<br>- 2nd experiment<br>- 3rd survey</td><td rowspan="4">(In Indonesia)<br>- Update Prototype<br>- 4th experiment<br>- 5th survey</td></tr>
<tr><td>06</td><td rowspan="4">Develop Prototype (for modelling)</td></tr>
<tr><td>07</td></tr>
<tr><td>08</td></tr>
<tr><td>09</td><td>Thesis Outline</td></tr>
<tr><td>10</td><td rowspan="3">Develop Prototype (metamodeling)</td><td>40-minute Seminar</td></tr>
<tr><td>11</td><td>Qualifying Dissertation</td><td rowspan="2">Thesis writing</td><td rowspan="2">Thesis Submission</td></tr>
<tr><td>12</td><td>Develop Prototype</td></tr>
</table>

# Chapter 4

# References


[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

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

[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] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

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

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

[20] 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.

[21] 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.

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

[23] 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.

[24] L. Schatzberg, "Applying bloom's and kolb's theories to teaching systems analysis and design," in *The Proceedings of ISECON*, vol. 19, 2002.

[25] 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.

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

# Chapter 5

# Publications

We have published papers in the following conferences or journals:

1. A. Yohannis, "Gamification of software modelling learning," in *the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016) Doctoral Symposium*. CEUR, 2016. [26].

# Appendix A

# 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 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. Regarding 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 skill sets 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 generation modifcation 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 the industry in the future I was interested in generating and transforming code automatically and using a formal model for the structure of the 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 benefit 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 sofware engineering 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 sometimes felt 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 made 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 is a structural issue with semantics, it is an abstract syntax issue. If it is something more peculiar, it is 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 a real project).
    - The best way to learn this was from experience which was gained by completing all the practicals.

8. **What were the challenges—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. However, actually understanding the metametamodel and all the ecore stuff seemed pointless. You do not need to understand what EClass, EEnum, EString etc. are. It is just unnecessary detail that's very specific to Eclipse and not 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.
    - sometimes 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 modelled in EMF, and the other can not, 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 would be helpful if there was more.

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

    - By ignoring them once I realised 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 many 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

# Appendix B

# Application of Deterding's Gameful Design Steps

## B.1 Strategy

### B.1.1 Define Target Outcome and Metrics

**Outcome**

- Students that use the gamified learning perform better than traditional ones in software modelling.

**Metrics**

- Scores that they get from solving given software modelling problems.
- Subjective opinions that learning through the gamified systems is better than the traditional ones.
- Model metrics of the models that they produce.

### B.1.2 Define Target Users, Context, Activities

**Users**

Computer Science Undergraduate Students with some knowledge of object-orientation, ideally from both programming and design, and a good understanding of software engineering

**Context**

One term of software modelling course (Time) and in the context of university software modelling course (Place)

#### Activities

Modelling, metamodelling, and model management

### B.1.3 Identify Constraints and Requirements

#### Laws and regulations

- Privacy regulation.

#### Scope (time, budget, personnel)

- Time. One term of software modelling.
- People. Student of software modelling-related courses.
- Budget. Available research budget.

#### Technological requirements

- Notebooks, computer desktops.
- Internet connection.

#### Others

- Align with the existing learning models and software modelling teaching best practices.

## B.2 Research

### B.2.1 Translate Users Activities into Behaviour Chains

#### Modelling

- Real-world problems are given using textual description (or videos, interviews, images, etc.)
- Perform abstraction/identify relevant concepts (objects, values, attributes, and operations) according to the requirements
- Translate them into classes and relationships
- Construct the model in the form of diagrams that represent the classes and relationships in different views (different diagrams)
- Evaluate the models

**Meta-Modelling**

- Models are given using textual description (or videos, interviews, images, figures, etc.)
- Perform abstraction/identify relevant concepts (classes, relationships, attributes) of the models
- Translate them into metamodel classes
- Construct metamodel diagrams that represent the metamodel
- Evaluate the metamodel

**Model Management**

- Problems are given using textual description (or videos, interviews, images, figures, etc.)
- Identify goals and constraints
- Determine the model management operations: Validation
- Transformation (model-to-model, text-to-model, model-to-text) Etc.
- Refine the operations into more detail steps
- Test
- Execute

### B.2.2 Identify User Needs, Motivations, challenges

**User needs**

- Master the software modelling.

**Motivations**

- Has competency in software modelling (Ability to solve problems and ability to produce artefacts or concrete products).
- Fun, challenging.
- Understanding the importance and advantages of software modelling.

**challenges**

- No fun, the presentation of software modelling is not interesting.
- Abstraction is difficult, choosing the most relevant elements according to the requirements.
- Heavy cognitive loads dealing with abstract concepts.

### B.2.3 Determine Gameful Design Fit

- Does the activity connect to an actual user need? **Yes**
- Is lacking motivation a central issue or opportunity (and not, e.g., poor usability)? **Both**
- Does the target activity involve an inherent challenge with a learnable skill? **Yes**
- Is affording experiences of competence the most effective and efcient way of improving motivation (and not, e.g., defusing fears)? **Yes**

## B.3 Synthesis

### B.3.1 Formulate Activity, Challenge, Motivation Triplets for Opportune Activities or Behaviours

**What motivations energize and direct the activity?**

Software modelling mastery: solving problems and build abstract/concrete artefacts

**What challenges are inherent in the activity?**

- Abstraction: determine the most relevant objects and relationships between them
- Diagramming: translate the resulting model into diagram
- Algorithmic operation: much like programming but for model management

**What challenges can be removed through automation or improving usability?**

- Simplify the real-world scenario to identify relevant objects
- Simplify the diagramming processes
- Simplify the algorithmic operations

**What challenges remain that the user can learn to get better at?**

Abstraction, diagramming, algorithmic operation

**What are the activities, challenges, and motives?**

- Activity: Construct model
- Challenge: abstraction, diagramming, algorithmic operation
- Motive: fear to make mistakes (-), solving problems (+), create artefacts (+)

## B.4 Ideation

### B.4.1 Brainstorming Ideas using Innovation Stems

**Challenge lenses**

Onboarding, scaffolded challenge, varied challenge

**Goal and motivation lenses**

Interim goals, viral calls to action, next base action, intrinsic rewards, secrets, templates, traces of others

**Action and object lenses**

Bite sized actions, interesting choices, limited choices, micro-flow, small pieces loosely joined, expressive objects, underdetermination, sensual objects

**Feedback lenses**

Immediate, juicy, actionable, appeal to motives, glanceable, varied, surprising, graspable progress.

**Example**

- Scaffolded challenge lens and Mastery motivation How might we spark a sense of mastery in abstraction, diagramming, and model operation?
- How might we use scaffolded challenge to make the abstraction, diagramming, and model operation more enjoyable?
- How might we spark a sense of mastery with scaffolded challenge?
- How might we alleviate fear of making mistakes with scaffolded challenges?

### B.4.2 Prioritise Ideas

Only one idea created so far. Learners are given a scaffolded series of problems to which they can exercise modelling, metamodelling, and model management. Motivating feedbacks to achieve mastery are integrated as well.

## B.5 Iterative Prototyping

Prototyping is still on going work.

# Appendix C

# Storyboard