Progress Report

Model-Driven Gamified Software Modelling Learning

Alfa Ryano Yohannis
ary506@york.ac.uk

Supervisors:
Dimitris Kolovos
Fiona Polack

Department of Computer Science
University of York
United Kingdom

May 5, 2017

Abstract

Motivated by the success of gameful approaches in different fields, this research harnesses the engaging nature of games combined with the effectiveness of pedagogy and the automation of Model-Driven Engineering to propose an environment for model-driven gamified software modelling learning. The environment allows tutors to create software modelling learning activities, and to generate software modelling learning games for learners to play. This report presents the motivation behind the environment and summarises the progress that has been made so far. This research also plans to complete the development of the environment's prototype, evaluate it, and officially release it to software modelling community in the next two and half years.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

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

1.1 Background

Software modelling is commonly perceived as a difficult subject since it requires a mastery of abstraction [2]. However, this subject has a significant role in software engineering education and practice. Successful application of software modelling requires skills in abstract modelling [3]i.e. being able to think abstractly about a system [4]. Thus, teaching modelling also means teaching abstraction [5]. Therefore, it is crucial to make students understand the value of abstraction [4]. Weak software modelling skills will cause software engineering students to face further challenges with their degrees, as most software engineering related subjects involve of inherent abstraction problems [6]. Drawing from multidisciplinary research to define an 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 [7]. Those processes are the common approaches applied in Computer Science, such as encapsulation and generalisation in object-oriented programming and algebraic simplification in symbolic computation. In the context of software engineering and computer science education, Kramer [6] and Hazzan [8] argued that abstraction is a central theme and a 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 [6].

> " ... software is an intangible object, and hence, requires highly developed cognitive skills for coping with different levels of abstraction." — Hazzan [8].

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 [9]. This 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 (regardless modelling can also be performed in textual manner e.g. Z language for formal modelling notations). However, grasping concepts represented by the visual notations and conversely presenting back the concepts into visual notations is not trivial, including presenting aspects of the concept in different visual notations as well as choosing the right levels of abstraction and moving between them. Learning and teaching these skills is challenging. To overcome the challenges, this research aims to develop a dedicated environment to support the learning of the modelling skills.

In recent years, the use of games and game elements, such as Gamification [10] and Serious Games [11], 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.

Real-world examples that show the success of the gamification are Duolingo[1] and Re-mission[2]. Duolingo is a gamified system of language learning. It embeds game elements, such as points, levels,and lives, to make language learning more fun. Re-mission is a third-person shooting game dedicated to young cancer patients and designed to teach and learn how to deal with cancer. The patients are invited to take part in an entertaining gameplay that will affect their specific behavioural and psychological outcomes producing effective cancer therapy.

Through a systematic review, Connolly et al. [12] studied the impact of computer games and serious games on engagement and learning in diverse fields. They reported that the majority of the studies presented empirical evidence about the positive impact of computer games and serious games. Using the same type of method, Hamari et al. [13] found that according to the majority of the reviewed papers, gamification does generate benefits and positive effects. Pedreira et al. [14] also performed a systematic literature review of the application of gamification in software engineering and it is found that most of the literature 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 research is at early stages. 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 aspects cannot be neglected. Several concepts from pedagogy will be applied to drive the design of the environment, particularly the best practices from instructional design, a mature field in designing learning

[1] https://www.duolingo.com/
[2] http://www.re-mission.net/

activities so that learning processes can be more efficient and effective. Therefore, incorporating gameful approaches as well as the best practices of instructional design is potential to tackle the problems in learning software modelling. The environment is being designed to be suitable for higher-level undergraduate and postgraduate students with some experience of software engineering.

The application of Model-Driven Engineering (MDE) approaches broadens opportunity to improve the meta-processes of the gamified software modelling learning. Instead of developing the software modelling games manually, a model-based approach is applied. An environment is being developed, and will facilitate the design of learning activities for topics in software modelling. The learning activities are then transformed to generate software modelling learning games.

1.2 Research Questions

This research proposes a hypothesis that "the use of gamified software modelling learning can improve learners' engagement and performance, compared to alternative methods such as class-based teaching". To generate such games, an environment needs to be developed. The environment has two main functionalities, it enables tutors to develop software modelling learning games, and facilitates students to learn software modelling through playing the games. The environment is expected can support various visual modelling notations and the creation of different games for learning software modelling. The word 'improve' implies that learning with gameful approaches enhances learners' engagement and learning performance. The learners 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, the following sub research questions need to be investigated:

1. Which game styles and elements, and pedagogical approaches are suitable for software modelling learning games?
2. Which of the aspects of developing a software modelling game can be automated through the use of model-driven engineering?
3. Do software modelling games improve learners' performance and engagement?

1.3 Research Objectives

The solution proposed by this research is to produce an environment 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:

1. Investigate game styles and elements that are suitable for software modelling games, and existing pedagogical approaches to optimise software modelling learning.

2. Design and develop a prototype of the environment that applies Model-Driven Engineering's best practices to automate software modelling learning game development.

3. Perform controlled experiments to measure the effectiveness of the environment in improving learning engagement and performance compared to traditional method—didactic learning without the support of gameful approaches.

1.4 Research Outputs

By the end of this research, two possible research outputs have identified will have been produced:

1. An environment for tutors to design and produce software modelling learning games and for learners to learn software modelling in gamified ways.

2. Results of controlled experiments—comparing learning engagement and performance of learners with and without software modelling games.

1.5 Research Scoping

In the beginning, the ambition of the research was to address software modelling in Model-Driven Engineering as a whole—comprising modelling, metamodelling, and model transformation. However, after consideration pf the available time, the scope has been restricted to focus on graphical software modelling. This research plans to perform literature study and develop a prototype in the first and second years and conduct experiments for validation in the third year respectively.

The remainder of this report is organised as follows. A detailed progress review is presented in Section 2 and a research plan in Section 3. Section 4 lists the publication of this research.

Chapter 2

Progress Review

In this section, the progress of this research is reviewed. First, the Design Science Research Methodology (DSRM) [1], the main research method employed in this research, is discussed briefly, and then followed by a discussion of progress on activities composing DSRM.

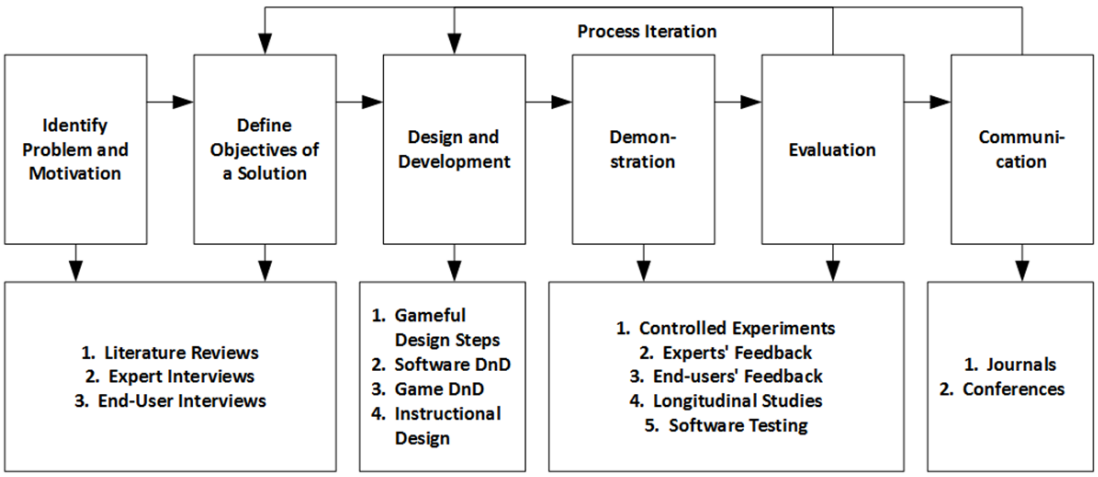


Figure 2.1: Design Science Research Methodology. Adapted from Peffer et al. [1].

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, an environment to design and generate software modelling learning games. DSRM provides a comprehensive conceptual environment 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, researchers can freely choose other more concrete research methods to carry out the activities. For example, literature reviews,

surveys, or expert interviews can be conducted 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 investigation have been widely presented in the Introduction chapter (Chapter 1). In this section, they have restated again in a succinct manner.

Software modelling plays a significant role in software engineering. However, its abstract nature makes it challenging to be mastered since most of its concepts are mostly presented in symbolical notations. To master software modelling, learners need to possess 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 can be difficult enough to demotivate learners from learning software modelling.

Apart from the problems mentioned above, the use of games or game elements rises as potential solutions to tackle motivational problems. Also, pedagogy has been a mature field to deliver effective learning processes. Therefore, incorporating gameful approaches as well as the pedagogy's best practices can bring significant benefits to tackle the problems in learning software modelling. An environment that unifies the gameful and pedagogical approaches is needed. Thus, this research is motivated to produce an environment that can support tutors to design gamified learning activities and learners to learn software modelling gamefully.

2.3 Define Requirements for a Software Modelling Learning Game Design Environment

The objectives for a solution proposed by this research are to develop an environment that can support tutors to develop software modelling learning games as well as learners to learn software modelling through playing the games, and to perform controlled experiments to measure the effectiveness of the environment in improving learning engagement and performance compared to traditional method (didactic learning without the support of gameful approaches).

The design of a solution should be based on existing foundations (models, theories, frameworks, etc.) [15]. Thus, some requirements have to be met to ensure the environment meets its desired functionality. Some requirements have been identified—pedagogical requirements and gaming requirements—and gathered from

literature review. These requirements provide guidance to the design process.

2.3.1 Pedagogical Requirements

Pedagogical requirements are classified into two categories, requirements from pedagogy-psychology literature and requirements from software modelling learning' best practices. They are presented in the following subsections.

Requirements from Pedagogy-Psychology Literature

Bloom's taxonomy is a model that categorises learning activities into six levels based on their cognitive load incrementally; the six levels are remember, understand, apply, analyse, evaluate, and create [16]. Performing activities in the *create* category is more challenging than activities in the lower categories since they demand high cognitive loads. All of these activities, whether they are arranged gradually or a combination of them, can give different variation and difficulty of a challenge to learners when they are performed in a learning context.

While learners are progressing in the environment, 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 Flow concept [17] can be applied. There are three challenges have been identified so far related to a pedagogical approach to control the degree of difficulty: 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 Flow theory. Concepts that are easier should be introduced earlier than the harder ones, and the difficulty should be gradually increased as learners progress. Likewise, application in the domains that are more familiar to learners should be introduced first and gradually shifted to the domains that are most unfamiliar. Combining these three dimensions–types of activities, concepts, and domains–could provide us with a variety of levels with different degrees of difficulties.

Motivation is an important aspect in the success of learning, and Keller's ARCS motivational model is also selected to address this aspect [18]. The model provides for each of its components—attention, relevance, confidence, satisfaction—a set of predefined techniques to maintain learners' motivation. In the course of a level of the environment, there are several activities. The ARCS' techniques can be applied to maintain learners' motivation along the course of completing a level. As an example, animations could be used to gain learners' attention, explaining the application of the concept being taught in the current level to give relevance, showing their progress in completing a level to maintain their confidence and to give them a reward after finishing a level for reward.

The concept of Scaffolding [19, 20] could also be applied to support learners. Throughout completing a level, scaffolding could be provided in several ways: reducing complex 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, provide help and

Table 2.1: Requirements derived from learning theories and models.

Category	Code	Requirements from Pedagogy-Psychology Literature
Pedagogical-Psychological Concepts	RM01	The environment facilitates the Bloom's taxonomy [16].
	RM02	The environment accomodates Kolb's experiential learning model [21].
	RM03	The environment allows Keller's ARCS motivational model [18].
	RM04	The environment enables scaffolded learning [19, 20].
	RM05	The environment supports the theory of Flow [17].

documentation, and give 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. Scaffolding is not generated automatically by the environment. Instead, tutors are given freedom to incorporate scaffolding into their learning activity design.

Kolb's experiential learning model [21] is also applied to the environment. The model proposes that knowledge is constructed through experience and is based on constructivism [21]. Kolb's model was selected since playing levels in the environment is similar to the learning cycle Kolb proposed. The cycle consists of 4 steps: concrete experience, reflective observation, abstract conceptualisation, and active experimentation.

So far, different concepts in pedagogy and psychology related to learning have been identified—from Bloom's taxonomy to Kolb's experiential learning. All of these concepts can provide principles that are useful to deliver effective learning. Thus, instead of rigorously design the environment only to certain concepts, it's better to allow tutors to apply these different concepts of learning into their learning activity design. Thus, this research aims to make the environment generic enough to accommodate different learning approaches. However, as a start, some concepts still have to be selected to guide and validate the design of the environment. Thus, the concepts presented in this subsection are also chosen as the requirements for the design of the environment (Table 2.1).

Requirements from Software Modelling Learning

Requirements from software modelling learning are the best practices pointed out by experts in the field about what and how software modelling should be taught and learnt. These requirements are used as key points to consider in the design phase of the software modelling learning games environment. The list of the identified requirements can be seen in Table 2.2.

Since the requirements in the list are vary and recommended from different experts, it is more generic if the environment can allow tutors to express these best practices into their learning activities based on their own preferences and needs

Table 2.2: Requirements derived from the literature review.

Category	Code	Environment's Requirements from Literature Review
Contents	RL01	Teach software modelling definition [22].
	RL02	Teach semantics, syntaxes, notations [22].
	RL03	Teach modelling and metamodelling [4, 23].
	RL04	Teach the application of software modelling [4, 24].
	RL05	Teach the engineering aspect of modelling [25].
	RL06	Teach modelling in various domains and contexts [22, 25].
Practices	RL07	Measure student's model's quality [26].
	RL08	Facilitate teaching problem solving first, detail specifications and tools get in the way [25].
	RL09	Provide support to solutions, not direct answers [25].
	RL10	Facilitate teaching with different modelling languages [4, 25].
	RL11	It is engaging (fun, challenging) [25].
	RL12	Teach from ground, real-world, familiar objects, up to abstraction [5].
Tool Design	RL13	Have adequate support and documentation [24].
	RL14	Designed to build knowledge incrementally [27].
	RL15	Provide flexibility to explore learning [27].
	RL16	Give positive reinforcement [27].
	RL17	Convince the learner about the value of the topic being learned [27].
	RL18	Have high usability [27].

rather than fixedly embed all the requirements into the environment.

2.3.2 Gaming Requirements

The main challenge addressed in this work is transforming software modelling learning into gameful activities. Given that there are different types of modelling languages (e.g. graphical, textual, projectional, tree-based) and types of games (continuous vs. level-based, real-time vs. turn-based, single-player vs. multiplayer), at this stage it is worth noting that our study is limited to level-based single-player games for languages with a 2D graphical syntax. For this style of game, the following concerns and challenges have been identified. These concerns and challenges are summarised as requirements and listed in Table 2.3.

Game Mechanics

Gameful design method [28] suggests that to apply gamification, the core activity should be identified before gameful modification can be implemented. Modelling using graphical modelling languages, such as constructing and modifying diagrams,

mainly involves design, editing, and planning activities. The LM-GM environment [29] mentions these activities as game mechanics that can be used for learning, specifically to address the *creating* category in Bloom's taxonomy [16]. Thus, the activity of constructing and modifying diagrams can be used as game mechanics as well, as it also involves design, editing, and planning activities as in design, editing, and planning game mechanics.

As for gameplay, learners are asked to construct or modify diagrams so that the diagrams become consistent with a given problem description and instruction. They do not always have to start with an empty canvas but could start from existing, incomplete diagrams that are pre-made to help them focus on the core concepts and activities being taught [28]. We name this kind of play as 'construction gameplay'. Other forms of play can be supported. For example, learners can be provided with a problem description and a number of diagrams reflecting candidate solutions and asked to select the correct solution(s). In another form, learners can be given a diagram and a number of statements in natural language and choose the proper interpretation(s) or statement(s). We call these two forms of play 'multiple-choice gameplay'. Tutors need to be able to define multiple levels of different gameplays and link them up to assemble complex games.

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

Assessing Correctness

There are several ways to assess consistency and correctness for the construction gameplay. One option is to require that the provided diagram is a 1:1 match with a reference solution given by the game designer. A more relaxed approach is only to require that the provided solution satisfies a number of constraints. Another option is to require some form of semantic equivalence between the two models (e.g. class diagrams that are consistent with an object diagram).

Rewards

Rewards are a major element of learning games, allowing users to develop a sense of achievement, show off their new skills, and compare themselves against other users, etc. So far, two types of rewards have been selected, positive reinforcement and achievements. Positive reinforcement can have the form of sounds, visual effects, or motivating messages to give feedback to players. Positive reinforcement is intended to improve the self-efficacy of players by letting them know whether their actions are in the right direction, so they can decide and continue to execute their next actions [30]. Another reward is a list of a learner's achievements, which can act like a 'CV' for a learner. It records the names, types, and numbers of levels and learning activities

Table 2.3: Gaming Requirements.

Category	Code	Gaming Requirements
Game Mechanics	GM01	support construction gameplay [29]
	GM02	support multiple-choice gameplay (select the best diagram)
	GM03	support multiple-choice gameplay (select the best interpretation)
Assessing Correctness	GC01	complete match validation
	GC02	partial match validation
	GC03	semantic equivalence validation
Rewards	GR01	positive reinforcement [30]
	GR02	list of achievements [30]

that they have been completed. Since a learning activity usually addresses a concept in software modelling, this could be used as a base to claim that the learners have acquired the competence – knowledge and skill – for that concept [30]. Building a learner's competence also means strengthening their intrinsic motivation [31] and make game playing activities important to them [32].

2.4 Design and Development

A web-based prototype of the environment has been developed to facilitate the design and implementation of software modelling games in line with the concerns discussed in the Pedagogical Requirements (Section 2.3.1) nad Gaming Requirements (Section 2.3.2). The prototype implements a metamodel-annotation-based approach for defining the concrete syntax of the targeted modelling languages. Metamodels are defined in Ecore [33] and their concrete syntax is specified using Eugenia-like annotations[1] [34]. Tutors can define a class as the root object of a metamodel (annotated with `@diagram`), denote classes to appears as nodes (`@node`) or links (`@link`), and denote a containment references to appear as compartments where elements of the type of the reference can be nested/spatially contained (`@compartment`). The instances of the annotated classes are displayed as visual elements in a web-based graphical editor built upon the MxGraph environment[2]. Tutors can adjust the appearance of the visual elements by setting the values of appropriate annotation parameters (for example, see Listing 2.1 and Figure 2.7-2.9 for the visual elements produced).

An annotated metamodel is then transformed using Epsilon's model-to-text transformation language (EGL) [35] and EMF [33] to generate its Java implementation code for further model management operations on the server, such as model validation and transformation (Figure 2.2). The transformation also pro-

[1] http://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial/
[2] https://jgraph.github.io/mxgraph/

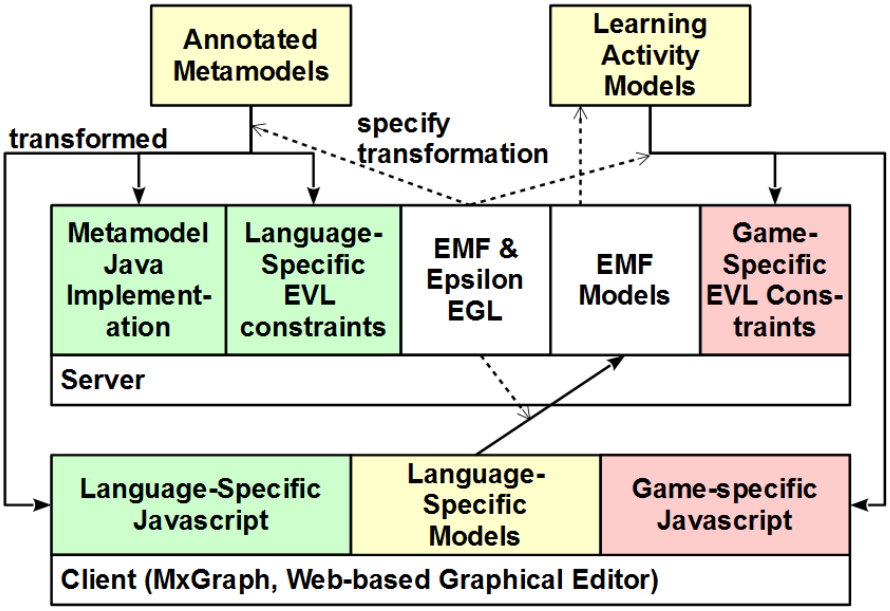


Figure 2.2: The environment's artefacts of the software modelling learning games.

duces Javascript that implements a language-specific web-based graphical editor built on top of the MxGraph environment. After the generation, tutors can create language-specific models using this editor and validate them using language-specific EVL [36] constraints (Figure 2.2). The models (the EMF models in Figure 2.2) can be used as base models in construction gameplay and are required as candidate solutions, and as diagrams to be selected and interpreted by learners in multiple-choice gameplay.

A modelling language has also been defined to design and link up different types of levels (see Figure 2.3 for its metamodel). We name the models constructed by this modelling language, *learning activity models* (for example, see Figure 2.5). Currently, elements that are supported by the learning activity modelling language are activity, objective, model, start, end, transition, and link. The activity element represents an activity in a learning process (in the transformation that follows each activity is transformed to a level). In an activity, tutors need to define *lesson* (description of the topic being taught), *instruction* (direction to complete the activity), the name of the target metamodel, and a set of *objectives*(elements to describe targets/conditions that learners have to meet in order to complete an activity).

A *model* is an element that refers to an existing model (the EMF models in Figure 2.2) created by tutors or a target model that will be produced by an activity. The referred model is used as the base model when learners start an activity, so they don't have to create the referred model from scratch. The *start* and *end* elements indicate where a learning activity should start and end. A *transition* element is a directed edge that connects one activity to another, and *link* element is also a directed edge that connects a model to an activity and *vice versa*. Facilitating tutors to create different learning activity models as well as to construct different models as base models for various visual modelling languages allows tutors t1o adjust the difficulty and support provided. This facility enables tutors to create various

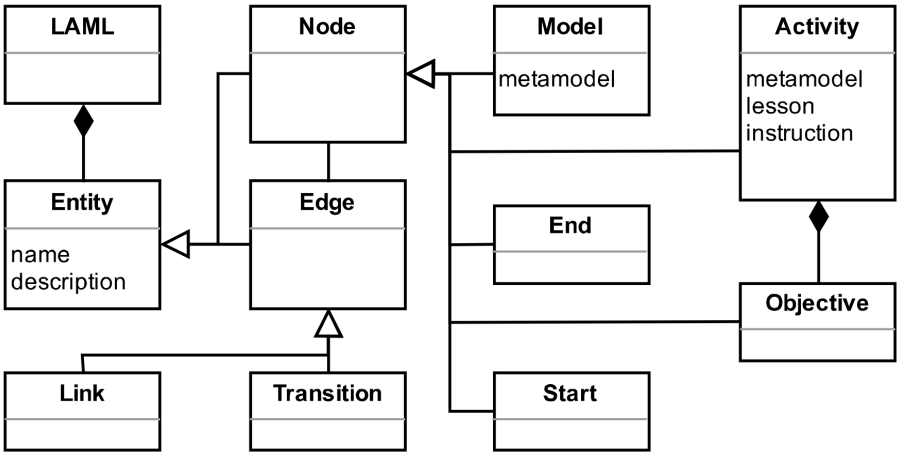


Figure 2.3: The class diagram of the *learning activity modelling language*.

learning contents and activities that accommodates different learning approaches. Thus, it addresses some requirements in Subsection 2.3.1.

Learning activity models are consumed by a model-to-text transformation to generate game-specific Javascript that implements a complete playable game (Figure 2.2). The environment can host a number of generated games and provides features such as authentication, progress management, learning activity inventory, model inventory, etc.

The generation also produces a game-specific EVL skeleton (Figure 2.2) for each level where tutors can define constraints and operations used for validation, to determine whether the level has been completed and all its objectives have been met. Currently, only the construction gameplay is supported at each level, and the correctness of solutions is assessed using the EVL constraints. So far with EVL, tutors can create rules to perform 1:1 match validation between a diagram and a reference solution and to check that a solution satisfies a number of constraints (assessing correctness requirement in Subsection 2.3.2 is addressed). The multiple-choice gameplay (Subsection 2.3.2), and other validation approaches (Subsection 2.3.2), as well as reward mechanisms (Subsection 2.3.2), are in development.

2.5 Demonstration and Evaluation

2.5.1 Demonstration

In this section, an example is presented to demonstrate the use of the environment in a game that introduces the concept of substates in UML statechart modelling. Learners are assumed to already understand the concepts of state, transition, start state, and final state. The following subsections present the different stages of the lifecycle of the game from definition to full use in game play.

Listing 2.1: A definition of statechart diagram using Emfatic and Eugenia-like annotations.

```
1  @namespace(uri="statechart", prefix="statechart")
2  package statechart;
3
4  @diagram
5  class Statechart {
6   val Entity[*] entities;
7  }
8  abstract class Entity {
9   attr String name = "";
10  attr String description = "";
11 }
12 abstract class Node extends Entity {
13  ref Edge[*]#source outgoing;
14  ref Edge[*]#target incoming;
15 }
16 abstract class Edge extends Entity {
17  ref Node#outgoing source;
18  ref Node#incoming target;
19 }
20 @link(label="name", source="source", target="target", label="name", endArrow="block
       ", blockendFill="1", endSize="6", width="120", height="120")
21 class Transition extends Edge {
22 }
23 @link(label="name", source="source", target="target", label="name", endArrow="none",
        blockendFill="1", endSize="6", width="120", height="120", dashed="1")
24 class Link extends Edge {
25 }
26 @node(label="name", shape="swimlane", childLayout= "stackLayout", collapsible="1",
       horizontalStack="0", resizeParent="0", resizeLast="1", rounded="1", marginBottom
       ="7", marginLeft="7", marginRight="7", marginTop="7", whiteSpace="wrap", width
       ="200", height="120", swimlaneFillColor="#FFFFFF")
27 class State extends Node {
28  @compartment(shape="swimlane", collapsible= "0", noLabel="1", editable="0",
        strokeColor="none", startSize="0")
29  val State[*] substates;
30 }
31 @node(label="description", shape="note", whiteSpace="wrap", width="200", height
       ="120")
32 class Note extends Node {
33 }
34 @node(label="name", shape="startState", whiteSpace="wrap", fillColor="#000000",
       width="30", height="30")
35 class Start extends Node {
36 }
37 @node(label="name", shape="endState", whiteSpace="wrap", fillColor="#FFFFFF", width
       ="30", height="30")
38 class End extends Node {
39 }
```

Define Statechart Metamodel

For the environment to support statechart modelling, designers need to define an annotated statechart metamodel as shown in Listing 2.1. In line 1 and 2, the namespace and package of the metamodel are defined by setting uri, prefix, and package to "statechart". Next, Statechart class is defined and annotated with `@diagram` to denote that the object of the Statechart class is the root object of the metamodel (the diagram). This class has a containment reference named 'entities' with of type Entity. Thus, all instances of Entity and its subclasses can be contained in a Statechart diagram (lines 4-7).

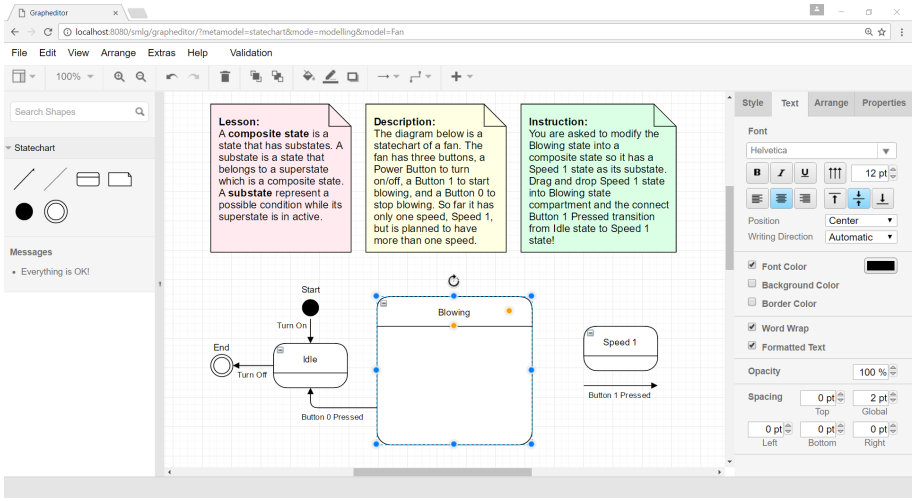


Figure 2.4: The MxGraph-based graphical editor to create statechart models.

The statechart diagram is intended to only support six basic elements: state, note, start, end, transition and link. Therefore, six classes are specified to represent the elements (lines 21, 24, 27, 32, 35, 38). State, note, start, and end elements are nodes in the diagram; their classes are derived from Node class, and each is annotated with `@node`, as well as its MxGraph-related attributes (fillColor, width, height, shape, etc.), to determine its appearance in the diagram (lines 26, 31, 34, 37). Similarly, transition and link are edges in the diagram, extend the Edge class and are annotated with `@link` (lines 20, 21, 23, 24). The State class has a containment reference named substates that also has type State class. This means that an instance of State class can also contain other instances of State class as its substates. The *substates* attribute is annotated with `@compartment` which means, in the diagram, a state has a container and other states can be placed inside it to become its substates (lines 28-29). The Node class has two references: outgoing and incoming. Both have the same type, Edge (lines 13-14). The Edge class also has two references: source and target. Both have the same type, Node class (lines 17-18).

Create Statechart Models

Using the generated graphical editor (Figure 2.4), designers can now create statechart models. The editor has a palette that contains statechart elements which they can drag and drop into a drawing area where they can arrange the elements to construct statechart models. The editor also has a property panel to modify the attributes and appearance of the models. Models that have been created can be saved and loaded again for further operations. The editor also can be used to create models as input/base models for learning activities.

Create Learning Activity Model

To teach a concept, tutors should design learning activities. In this example, a learning activity model to learn the concept of substates is created (Figure 2.5).

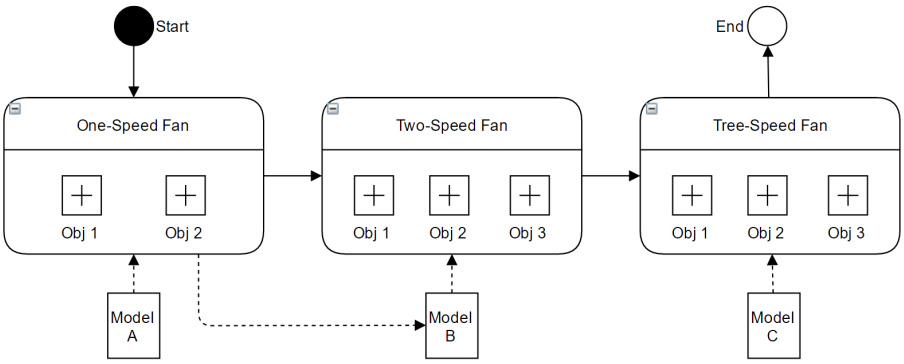


Figure 2.5: A learning activity flow for learning the concept of Substates in state-machine modelling.

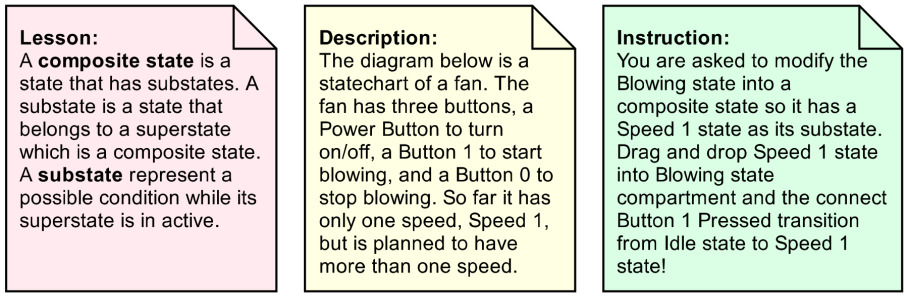
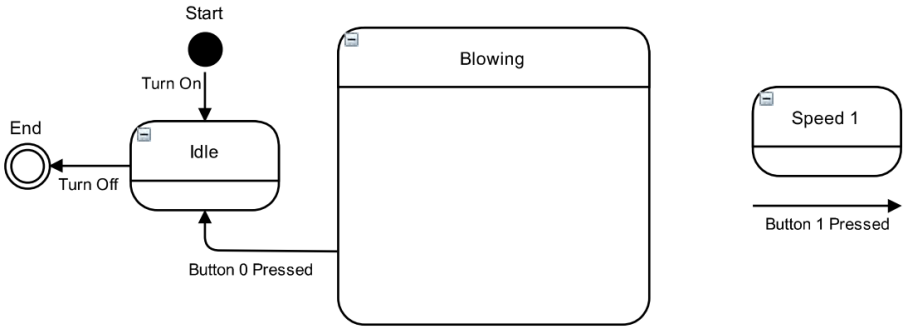


Figure 2.6: Lesson, model description, and instruction.

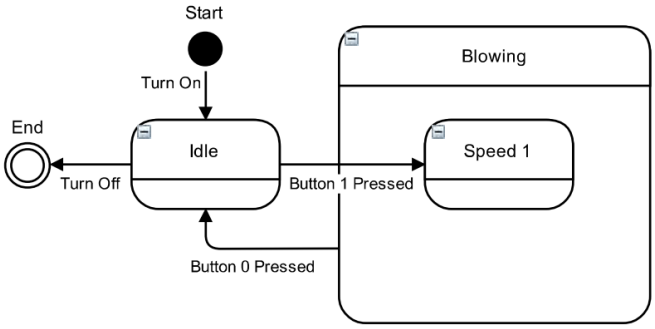
The case is an electrical fan that has two main states, Idle and Blowing. Learners will be asked to modify the Blowing state, so it becomes a composite state with substates that represent the different speeds of blowing. The learning activity has three activities, namely One-Speed Fan, Two-Speed Fan, and Three-Speed Fan. The activities are arranged *per se* to accommodate Flow state [17] so learners can progress from the easiest activity to the hardest one. The activities are explained in more detail in the following subsections.

Each activity has a lesson and instruction properties. Lesson contains an explanation of the concepts to be taught; instruction contains commands or questions that learners need to execute or answer. Learners must meet all the objectives of an activity to move to the next activity. Each activity can consume existing models (Model A, B, and C in Figure 2.5) as its base models – so that learners do not have to create models from scratch every time – and produce a model (Model B in Figure 2.5) to be used in its next activity. Each model has a description property, which helps learners to understand about the model. An example of the lesson, model description, and instruction properties of the One-Speed Fan activity (Figure 2.5) is displayed in Figure 2.6.

One-Speed Fan Activity. In this activity (Figure 2.7), learners are introduced to one substate only. The activity starts with a base model, and learners are required to modify the base model (Figure 2.7a) to match the target model (Figure 2.7b). The base model corresponds to Model A in Figure 2.5, which refers to an existing base model and will be loaded once the activity is executed, so learners do



(a) Base model



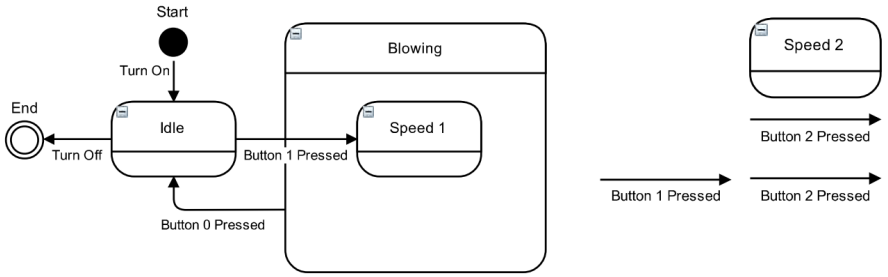
(b) Target model

Figure 2.7: The One-Speed Fan example.

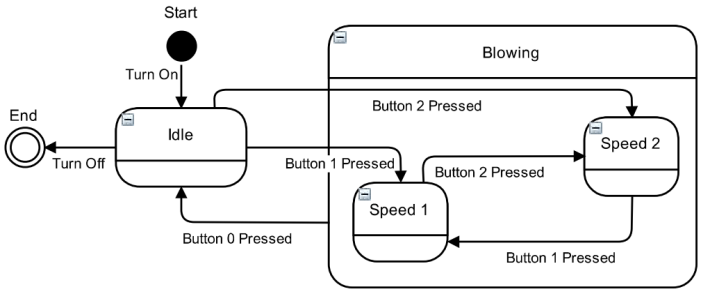
not need to create the model from the start.

The example case in this activity is a fan that has three buttons, a Power Button to turn on/off, a Button 1 to start blowing, and a Button 0 to stop blowing. So far it has only one speed, Speed 1, but is planned to have more than one speed. Learners are asked to modify the Blowing state in Figure 2.7a into a composite state by moving the Speed 1 state into the Blowing state compartment and connecting the Button 1 Pressed transition from the Idle state to the Speed 1 state. Since in Figure 2.5 this activity is designed to have only two objectives, the two objectives are adjusted and defined as follow: Objective 1 "The Blowing state contains the Speed 1 substate" and Objective 2 "Button 1 Pressed transition connects the Idle state to the Speed 1 substate".

Two-Speed Fan Activity. The Two-Speed Fan activity (Figure 2.8) consumes the model that has been produced in the first activity (Model B in Figure 2.5). Any model generated in the previous activity becomes the base model for modelling in this activity. In accordance with the Flow concept [17], this second activity should be more challenging. Therefore, the activity (Figure 2.8) challenges learners with one additional state and three new transitions. Now the case has changed. The fan has an extra button, Button 2, to support 2-speed blowing. When Button 1 is pressed, the fan blows in speed 1 and when Button 2 is pressed, the fan blows in speed 2. Thus, learners need to modify the Blowing state into a composite state, so that it has two-speed states. The fan can move from the Idle state to the Speed 1 state, from the Idle state to the Speed 2 state, and from the Speed 1 state to the

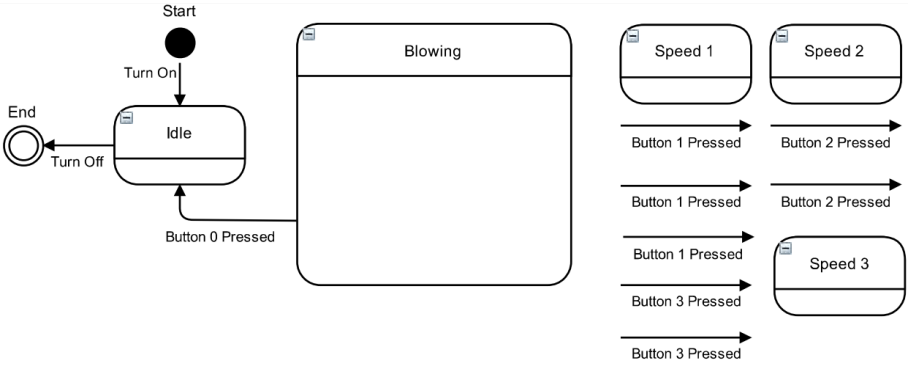


(a) Base model

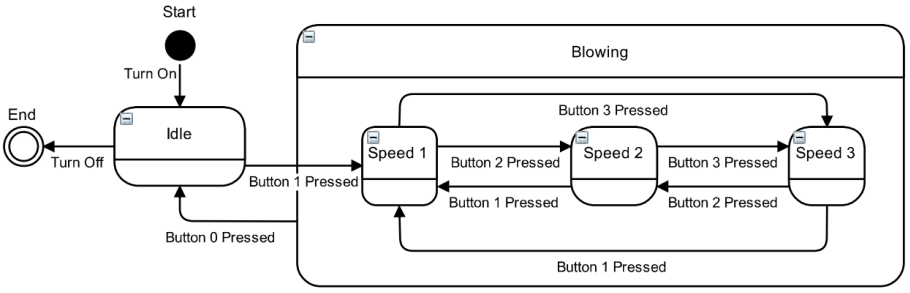


(b) Target model

Figure 2.8: The Two-Speed Fan example.



(a) Base model



(b) Target model

Figure 2.9: The Three-Speed Fan example.

Speed 2 state or *vice versa*.

Three-Speed Fan Activity. The Three-Speed Fan activity (Figure 2.9) should be harder than the second activity. The fan now supports 3 speeds of blowing,

Listing 2.2: Validation template for objectives in One-Speed Fan activity/level.

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

Listing 2.3: Validation realisation for Objective 1 in One-Speed Fan activity/level.

```
context Statechart {
 constraint obj_1 {
  check:
   self.obj_1()
  message:
   "FAIL: Blowing state contains Speed 1 substate"
 }
 ...
}
operation Statechart obj_1(): Boolean {
 return State.all.exists(state | state.name = "Blowing" and state.substates.exists(
     substate | substate.name = "Speed 1"))
}
...
```

but it has been modified so it cannot go directly to Speed 2 or Speed 3 without firstly going to a state with a lower speed. In other words, the transition from Idle can only go to Speed 1 for the fan to start blowing. Thus, starting from the model C as the base model (Figure 2.5), learners are required to modify the Blowing state into a composite state with 3 speeds of blowing and to only allow a transition from Idle to Speed 1.

Generate Game

After constructing the learning activity (Figure 2.5), designers can now generate a path of levels (or stages). Each generated level corresponds to an activity defined in the learning activity and has an EVL template for validation. The EVL template is shown in Listing 2.2 which can be extended by designers to write constraints that fit the level scenarios and objectives. The number of constraints and operations in the template corresponds to the number of objectives defined in the level's learning activity activity in Figure 2.5. As an example, implementation of validation, Listing 2.3 shows the EVL code to check whether the Blowing state has a Speed 1 substate, as intended by Objective 1 in One-Speed activity.

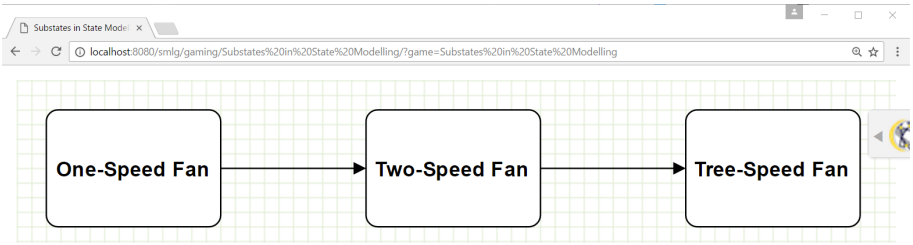


Figure 2.10: The path for learning substate concept with its levels.

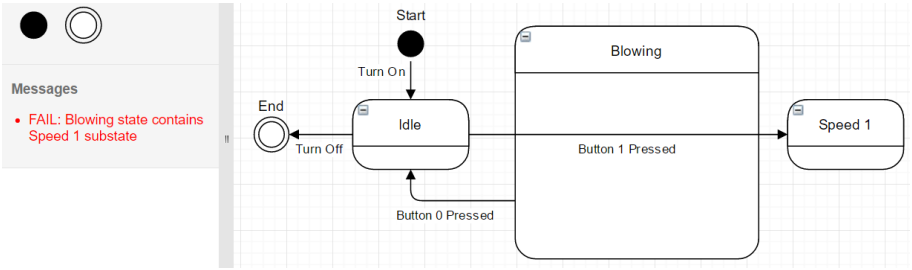


Figure 2.11: A message is displayed to learners indicating the objective that has not been fulfilled.

Play Game

After generating the learning activity into a path of ordered levels and defining its levels' validation, learners can choose the path and play its levels as depicted in Figure 2.10. When learners choose the first level, an MxGraph-based editor is displayed, and the lesson, model description, instruction, objectives, and base model are presented to learners. Learners can then start to modify the base model to reach the target model.

Every attempt to change the model triggers an operation to validate the current model using the defined EVL constraints to assess whether the model has met the current level's objectives. If not, feedback is displayed to learners indicating the objectives that have not been satisfied. For example, in Figure 2.11, the Speed 1 substate has not been put into the Blowing state even though the Button 1 Pressed transition has connected the Idle state to the Speed 1 substate. If all objectives have been fulfilled, the level is considered completed, an appropriate message is produced to provide positive reinforcement, and the learner can proceed to the next level.

2.5.2 Evaluation Plan

Two forms of assessment have been planned to evaluate the environment. The first assessment is to measure the effectiveness and engagement of modelling games vs. class-based lecturing using controlled experiments involving undergraduate and postgraduate students. Subjects will be grouped into two groups, an experimental group, and a control group. While the control group will only learn from attending class-based lecturing, the other group will also learn using games. Both will be tested with relevant problems before and after teaching to measure their im-

provement. The gap between the two group will be used to measure the effect of the games. For measuring the engagement, the User Engagement Scale [37] will be used. Even though this evaluation is feasible since an adequate number of software engineering students are available, challenges still exist. The quality of the class-based lecturing, the problems given in the tests, and the games should be consistent, ensuring that they carefully designed to address the same educational objectives. The other challenge is to have the two groups of subjects have balanced quality and distribution.

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

2.6 Communication

The progress of this research should be communicated to a larger audience. So far, two papers have been prepared for publication. The first paper [38] has been published (A. Yohannis, Gamication of software modelling learning, in *the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016) Doctoral Symposium*. CEUR, 2016) in a doctoral symposium, and the other one is already submitted to a conference and under review. The details of the two publications can be found in the Publications section (Chapter 4).

Chapter 3

Research Plan

Table 3.1: Research Timetable

<table>
<tr><th>Month</th><th>2017</th><th>2018</th><th>2019</th></tr>
<tr><td>1</td><td rowspan="3">Develop Prototype</td><td rowspan="3">- Thesis Writing
- Update Prototype</td><td rowspan="3">- Release Framework
- Thesis Writing</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td><td>Progress Report &
PhD Confirmation</td><td>Thesis Audit 1</td><td>40-Minute Seminar</td></tr>
<tr><td>5</td><td rowspan="4">Develop Prototype</td><td>- Update Prototype
- Thesis Writing</td><td rowspan="2">Thesis Writing</td></tr>
<tr><td>6</td><td>Experiment and Survey in Indonesia</td></tr>
<tr><td>7</td><td rowspan="2">- Update Prototype
- Thesis Writing</td><td>Thesis Submission</td></tr>
<tr><td>8</td><td rowspan="4"></td></tr>
<tr><td>9</td><td>Thesis Outline</td><td>Thesis Audit 2</td></tr>
<tr><td>10</td><td rowspan="3">- Experiment and Survey in UK
- Update Prototype</td><td rowspan="3">- Evaluate Prototype
- Design Formal Release
- Thesis Writing</td></tr>
<tr><td>11</td></tr>
<tr><td>12</td></tr>
</table>

For the next two and half years, this research plans to execute these following actions. The research timetable is displayed in Table 3.1.

- **May-Aug 2017**: Further develop and test the prototype. In particular, this research plans to extend the diversity of modelling languages that the prototype can accommodate by extending the list of supported annotations, as well as add multiple choice gameplay, semantic equivalence validation, and reward mechanisms. Also, two facilities will be added: a fine-grained logging facility that will allow the environment to record learner-game interaction for further analysis and a facility that monitors the progress of learners.
- **May-Aug 2017**: Prepare the required Thesis Outline.
- **Oct 2017-Dec 2017**: This research will prepare and conduct experiment in York to execute the evaluation plan in Subsection 2.5.2 and then update the prototype based on observations from the experiment.
- **Jan-Mar 2018**: Start writing thesis report based on Thesis Outline and update the prototype.

- **Apr 2018**: Conduct Thesis Audit 1 and prepare the experiment that will be executed in Indonesia.
- **May 2018**: Update thesis report and update prototype based on Thesis Audit 1, and prepare the experiment that will be executed in Indonesia.
- **Jun 2018**: This research will conduct an experiment in Indonesia to execute the evaluation plan in Subsection 2.5.2.
- **Jul-Aug 2018**: Update thesis report and prototype based on the experiment in Indonesia.
- **Sep 2018**: Conduct Thesis Audit 2.
- **Oct-Dec 2018**: Update the thesis report and prototype based on Thesis Audit 2. Design a formal release since the environment is also planned to be released formally to software modelling community to gain recognition and feedback.
- **Jan-March 2019**: Release the environment to software modelling community and update thesis report.
- **Apr 2019**: Prepare and conduct 40-Minute Seminar.
- **May-June 2019**: Update thesis report based on 40-Minute Seminar and for Thesis Submission.
- **Jul 2019**: Perform Thesis Submission.
- **Aug-Dec 2019**: Reserved time.

Chapter 4

Publications

Papers have been published or are under review 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. [38].

2. A. Yohannis, D. Kolovos, and F. Polack, "Towards Model-Driven Gamified Software Modelling Learning," in *the ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017)*. (under review).

Bibliography

[1] 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.

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

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

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

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

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

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

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

[9] 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.

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

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

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

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

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

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

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

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

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

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

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

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

[34] 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.

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

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

[37] E. N. Wiebe, A. Lamb, M. Hardy, and D. Sharek, "Measuring engagement in video game-based environments: Investigation of the user engagement scale," *Computers in Human Behavior*, vol. 32, pp. 123–132, 2014.

[38] 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.