Qualifying Dissertation

Gamified Software Modelling Learning

Alfa Ryano Yohannis
ary506@york.ac.uk

Supervisor:
Dimitris Kolovos
Fiona Polack

Department of Computer Science
University of York
United Kingdom

October 28, 2016

Abstract

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

Contents

Chapter 1

Introduction

Software modelling is commonly perceived as a difficult subject since it requires a mastery of abstraction [1]. However, this subject has a fundamental and crucial role in software engineering education and practice. Failure to master this topic will affect the students abstraction capability which is essential for analysing and designing real-world software. 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]. In the context of computer science and software engineering education, Kramer and Hazzan argued that abstraction is the central theme or key skill for computing [5, 6].

> "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 [6].

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 [7]. Abstraction also requires students to grasp skills in information hiding, generalisation, approximation or reformulation and separating relevant from irrelevant aspects [8]. To overcome these challenges, we need to put more effort into software modelling learning design, developing a more concrete and motivating presentation which can engage students and facilitate deep learning.

In the recent years, the use of games and game elements for purposes other than leisure has drawn significant attention. Gamification [9] and Serious Games

[10] (further we use 'Gamified Approach' term to comprise both concepts) have been proposed as solutions to motivational problems that emerge when users are required to engage in activities that they perceive as boring, irrelevant, or difficult. Figure 1.1 shows us the growth of "Gamification" on Google Trends[1] since February 2010.

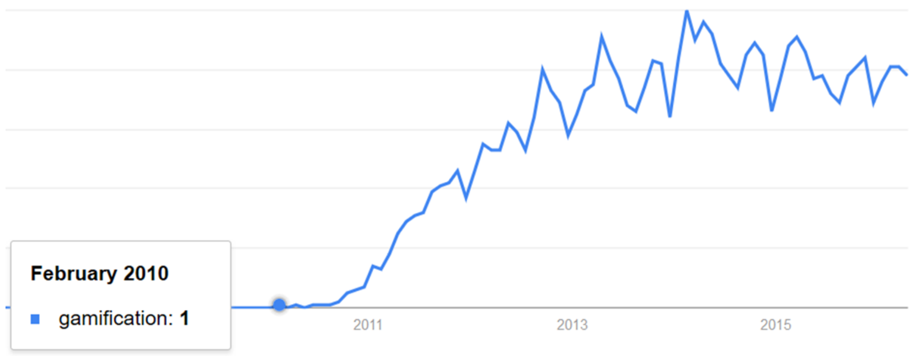


Figure 1.1: "Gamification" on Google Trends per 12 April 2016.

Real-world examples that show the success of the Gamified Approach is 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 involve in an entertaining gameplay that will affect their specific behavioural and psychological outcomes producing effective cancer therapy.

Through a systematic review, Connolly et al. 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 [11]. Using the same type of method, Hamari et al. found that according to the majority of the reviewed papers, gamification does generate benefits and positive effects [12]. Specifically in the field of software engineering, Pedreira el al. 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 of their work. They argued that existing studies in the field be quite new, thus more research effort is needed on investigating the impact of gamication in software engineering [13].

The report of the positive impact of Gamified Approach in various fields encourages us to apply the Gamified Approach in software modelling learning, a subject that has received little attention so far. This situation broadens our opportunity to produce novel research contribution regarding the application of gamification in the field of software modelling.

Instead of developing the software modelling games manually, we plan to fol-

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

low a model-based approach. We will develop a design framework for the games, that will systematically and semi-automatically drive gamification design to produce software modelling learning games. Essentially, our study lies in the intersection between software modelling, learning, and games as depicted in Figure 1.2. Therefore, pedagogical aspect cannot be neglected. We will apply several learning models from pedagogy to drive the design of our learning game.

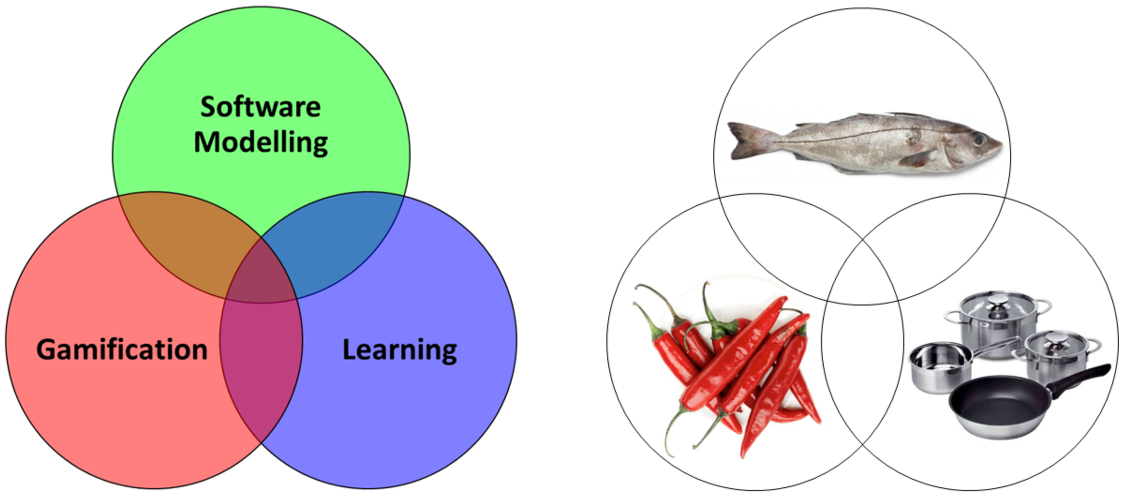


Figure 1.2: How to 'Cook' a Gameful Software Modelling Learning?

The remainder of this report is organised as follows. We provide a detailed literature review in Section 2 and propose the research proposal in Section 3. Finally, in Section 4, we end this qualifying dissertation with a discussion on our preliminary results.

Chapter 2

Field Survey and Review

Since gamification of software modelling learning is a multidisciplinary research topic, the literature review categorised is organised by the contributing fields, including the subtopics resulting from the interaction between them. So far, we have identified three major contributing fields, software modelling, ludology, and pedagogy (Figure 2.1). They will be discussed below, and a brief review of the research methodology concludes this chapter.

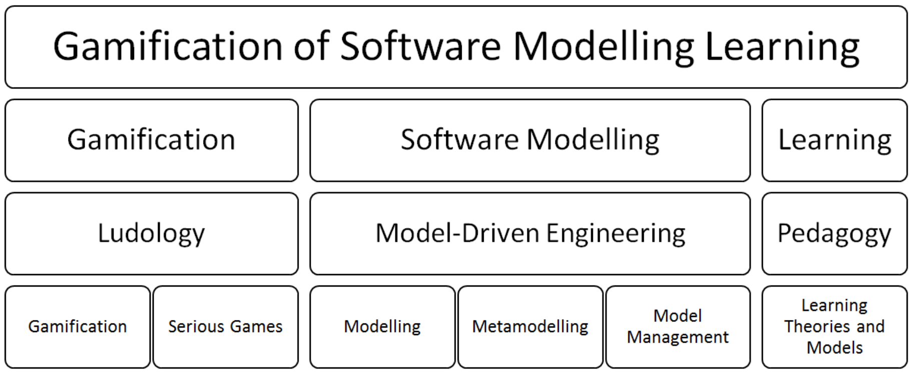


Figure 2.1: Contributing fields to the gamified software modelling learning.

2.1 Pedagogy

Designing the gamification of software modelling learning cannot be separated from the field of pedagogy since the core process that will be supported by the gamification is a learning process. There is an abundance of literature in the field of pedagogy how we can understand learning processes and from it, we could draw principles that will guide the design of software modelling learning games. We have selected five learning models. The models are selected because each of them has unique characteristics that we considered will address certain design concerns of the gamification. They are discussed briefly in the following subsections.

2.1.1 Csikszentmihalyi's Flow Theory

Csikszentmihalyi proposed the theory of Flow which states that to maintain learners' engagement in an activity, the given challenges and their competence have to be kept balanced [14]. Once the challenges are too difficult, they will arrive at a state of anxiety that demotivate them and then forces them to withdraw from the activity. On the other side, if the challenges are too easy and his competence is very advanced, he will be in the state of boredom which also makes it likely for him to withdraw from the activity (Figure 2.2).

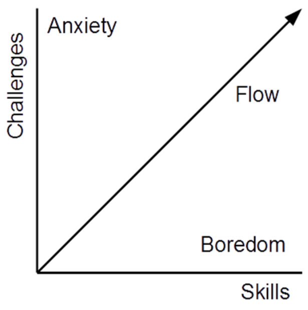


Figure 2.2: Csikszentmihalyi's Flow Theory [14].

This theory has been applied in many fields, such as education, training, and games to keep participants engaged with systems. Specifically for games, flow theory influences the design of levels. The given challenges have to be adjusted so they match the skills of players at every stage. Chen commented that well-designed games should bring their players to the flow states, balanced states between challenges and skills, delivering happiness and pleasure [15]. In the field of education, Liao also used the theory as a framework to study the emotional and cognitive responses to distance learning systems. He found out that flow theory, the cause and effect of the flow experience, also works when students use distance learning systems [16]. We plan to use the theory to guide the design of levels, balancing challenges and learners' competences.

2.1.2 Zone of Proximal Development/Scaffolding

Zone of Proximal Development (ZPD) was proposed by Lev Vygotsky in the context of adolescent development [17]. This term was also called Scaffolding by Wood et al. in the context of learning [18]. The theory states that during a learning process, learners have to be reinforced, particularly for knowledge or skills that are tough to learn without the support of others, to develop their competence until they can acquire the knowledge and skills on their own (Figure 2.3).

Kao et al. studied customised scaffolds of an educational game for learning physics [19] and Tsai et al. investigated the use of scaffolding aids with importance and use of targeted content knowledge in educational simulation games [20]. Both found proper scaffolding facilitates learning and helps learners to have better performance. Since this research also addresses learning, we plan that the game should also provide scaffolding to learners. It should present helpers and cues to learners

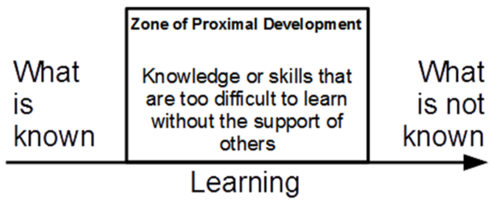

Zone of Proximal Development
What is known
Knowledge or skills that are too difficult to learn without the support of others
What is not known
Learning

Figure 2.3: Zone of Proximal Development/Scaffolding [17],[18].

to help them solve problems and grasp important concepts. The scaffolding will be removed gradually as the competence of the learners grows.

2.1.3 Revised Bloom's Taxonomy

Bloom's Taxonomy [21] is a framework of cognitive levels for the learning process. The framework has been proven very useful for more than 50 years, all over the world, in supporting educators design learning processes [22]. It consists of six activities for learners, namely remember, understand, apply, analyse, evaluate, and create, and order them according to their cognitive load levels with 'remember' at the bottom and 'create' as the activity that requires the highest cognitive load (Figure 2.4).

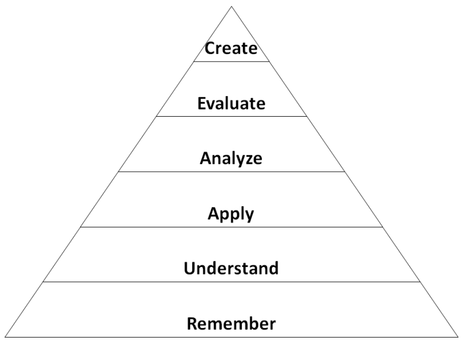


Figure 2.4: Revised Bloom's Taxonomy[21].

Bloom's taxonomy has been used by other researchers in their studies. Arnab et al. utilised learning activities in the taxonomy to identify and map learning mechanics to game mechanics to construct a Learning Mechanic-Game Mechanics model for designing Serious Games [23]. Von Vangenheim et al. [24] also developed an educational game for teaching SCRUM and applied activities in the taxonomy as a standard to assess the achievement of their respondents. Bloom's Taxonomy will give us a variety of options—activities and challenges—in designing the levels of our game.

2.1.4 Kolb's Experiential Learning Model

Kolb's Experiential Learning Model is a model of teaching and learning through experience and reflection on actions [25]. It states that knowledge development is a product of experience or iterative search. The model has a cycle consisting of four steps: abstraction conceptualisation, active experimentation, concrete experience, and reflective observation (Figure 2.5).

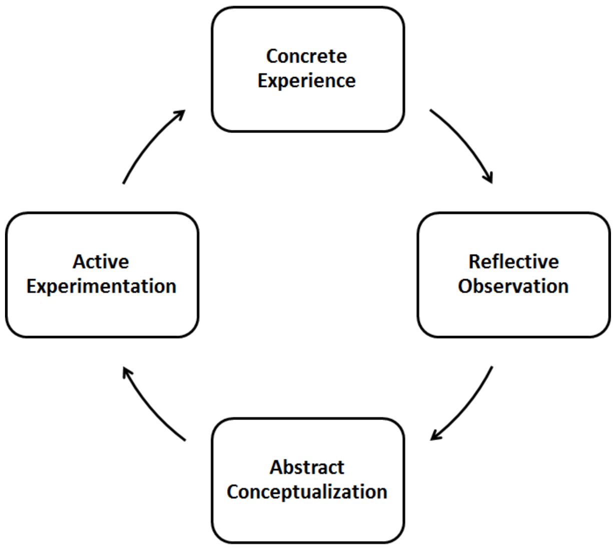


Figure 2.5: Kolb's Experiential Learning Model [25].

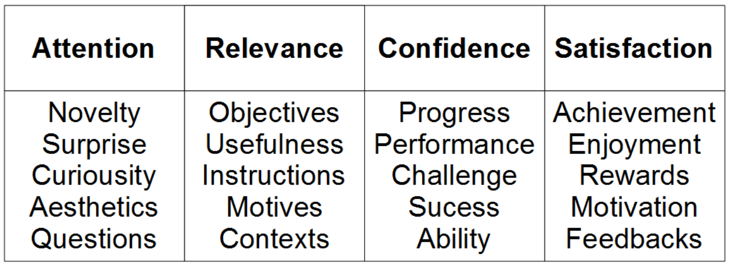
As an example, in the case of didactic learning, learners are firstly taught about the theory. In this step, learners perform abstract conceptualisation. They then move to practical activities in which they are asked to carry out some active experimentation of the theory they were taught in the class. Through the experimentation, they obtain concrete experience. The results might be to some degree confirm to or contrast the theory. In the end, through reflective observation, the results are then reconciled with the knowledge they had acquired based on the theory, whether the knowledge is updated or not.

A game is a potential tool to realise the experiential learning Kolb proposed, since playing games demands learners to actively participate in their course, reflect on their achievements as well as strategies and outcomes, and plan their next actions carefully to improve their performance. As a showcase, Bliemel and Hassan used an IT manager-like game as an experiential learning tool and found that the experience developed during playing the game helps learners to connect IT management theories and their applications [26]. Boctor studied the use of a game that facilitates experiential learning to reinforce the learning of nursing material. His respondents found this learning method beneficial and enjoyable [27]. Kolb's cycle provide us a framework how we should design activity cycle when learners engage in learning processes and also help the learners to interconnect their knowledge and real experiences.

2.1.5 Keller's ARCS Motivational Learning Model

Keller's Motivational Model is a set of steps for promoting and maintaining motivation in a learning process [28]. The steps are attention, relevance, confidence, and satisfaction (Figure 2.6). Attention means how to arouse learners to get their focus and awareness. Relevance is using examples and languages familiar to learners. Confidence means establishing and maintaining a positive attitude toward completing a learning process. Satisfaction is achieved when the learners' achievements are valued or rewarded.

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



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

Figure 2.6: Keller's Motivational Model [28].

Throughout their experiment using the ARCS model to examine the motivational aspect when learners play digital game-based learning, Huang et al. found that motivational processing (attention, relevance, and confidence) need to be considered when designing digital game-based learning [29]. Likewise, Derbali and Frasson studied players' motivation while playing a serious game. Using Keller's motivational model and electroencephalography to measure learners' motivation, they found out that learning using serious game significantly increases learner motivation [30]. Therefore, for the design of our game, we judged Keller's model is useful for designing the flow of a level of a software modelling learning game as it mentions steps that we need to focus on maintaining learners' motivation. We will implement the four steps of Keller's model in the course of a level.

In this section, we have presented learning models that we plan to apply in our design. The learning models presented include the theory of Flow, Scaffolding, Bloom's taxonomy, experiential learning, and motivational learning. Each model has their contributions to the design of our game. Theory of Flow tells us to balance challenges with learners' competency. Scaffolding suggests us to give reinforcement to learners to help them grasp difficult but important concepts. Bloom's taxonomy provides us six activities with different cognitive loads to design activities and challenges in a level. Experiential learning gives us a framework to design activity cycle for learner-game interaction to support learners make the connection between their knowledge and real experience. The motivational model provides us four components that we should consider in our level design to maintain learners' motivation. We will elaborate and discuss these learning models and our design in more detail in section 4.3.

2.2 Ludology

Likewise pedagogy, ludology is of significant relevance to our work. Concepts that are from Ludology, such as serious games, gamification, and gameful design steps [31] will be discussed briefly in the following sections.

2.2.1 Serious Games and Gamification

Through this report, two terms related to games often appear, serious games and gamification. Abs defines serious games “games that have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement” [32], whereas Deterding et al. define gamification as “the use of game design elements in non-game contexts” [9]. While both are used for purposes other than leisure, there is noticeable difference. Serious games are full-fledged games—they have all characteristics to be classified as games. In contrast, gamification focuses on the use of game elements, not on a development of a complete game.

Other confusion is on the word 'gamification'. The word 'gamification' could also be defined as a process [33]. The process is adding game elements to something that is originally not a game, so it becomes more like a game, but there is a moment—a threshold—when we keep adding game elements, it has all the characteristics to be defined as a game—it satisfies the definition of a game [34]. In other words, gamification process could also transform an entity into a game, but the entity could still own its original characteristics.

Therefore, to avoid the confusion throughout this report, we clarify that when we use the word 'game', it could refer to two meanings. First, it relates to the common popular definition of a game, and second, it refers to the product of a gamification process. Likewise, the word 'gamification' could refer to two definitions. First, it is a process—integrating game elements into an entity—and second, it is the product of the process after integrating game elements.

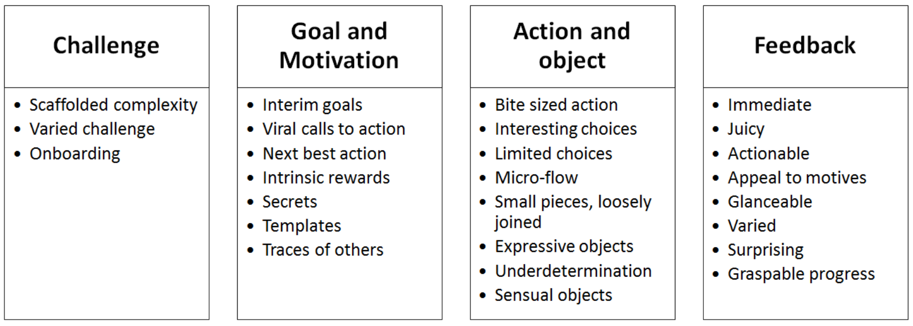
2.2.2 Gameful Design Method

There are several existing game and gamification design-related frameworks that could be utilised to organise the integration of game elements into a learning process. Some of them are specific in pedagogical context and depends on the use of full-fledged games [35, 36, 37]. Others act as an inventory of problem-solution mappings where designers could choose one or more game mechanics that are appropriate to their problems [23], [38]. The rest are very general procedural steps and raised from non-pedagogical context [39, 40].

However, all of them are very high level conceptually and less in provoking genuine gameful design. Genuine means the design should arise from target users through a deeply reflective and innovative process about a problem; it is core activity that wants the be gamified, and it is gameful solution. Consequently, the gameful design should not be limited to the adoption of mechanics from a fixed inventory, but should be stimulating enough to produce new forms of mechanics that fit with the stakeholder-specific aspects. The 'genuine' also imply that the framework should be generic, not limited to certain domains and could be applied across domains.

Depart from those thoughts, we have identified Deterding's Gameful Design framework [31] as a framework that could facilitate our consideration. Moreover, the framework has the following characteristics: (1) it bases its solution on target users' context (users, context, activities), (2) it focuses on the core activity that needs to be gamified, (3) it is concerned with the challenges and motivation that are inherent to the core activity, (4) it provokes ideation of a genuine gameful design, not just limited to the a collection of existing solutions (the solutions are used as a starting point to stimulate ideation), (5) it supports iterative prototyping, (6) it is generic and applicable across domains.

The Gameful Design framework consists of design lenses, skill atoms, and prescriptive guidelines. Design lenses are derived from game elements. A design lens is a combination of an easy-to-remember name, a short statement of a design principle, and a set of related questions that help the designer to view activities, motivation, and challenges from the perspective of game element that the lens represents. Deterding's Design Lenses comes with four categories of lenses: challenge, goal and motivation, action and object, and feedback. Each category comes with lenses which can be utilised to view activities, challenges, and motivation from game element perspective to generate innovative ideas of gamification (Figure 2.7).



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

Figure 2.7: Deterding's design lenses [31].

A skill atom describes a feedback loop between a user and a that is organised around a central challenge or skill (Figure 2.8). At first, the user takes actions driven by his motivations and goals, which forms inputs to the system's rule engine that determines changes of states of the system. The system returns feedback to the user, which then will be integrated into the user's mental model of the system. Through repeated action-feedback cycles, the user masters the intended skill.

The prescriptive guidelines comprise five phases: strategy, research, synthesis,

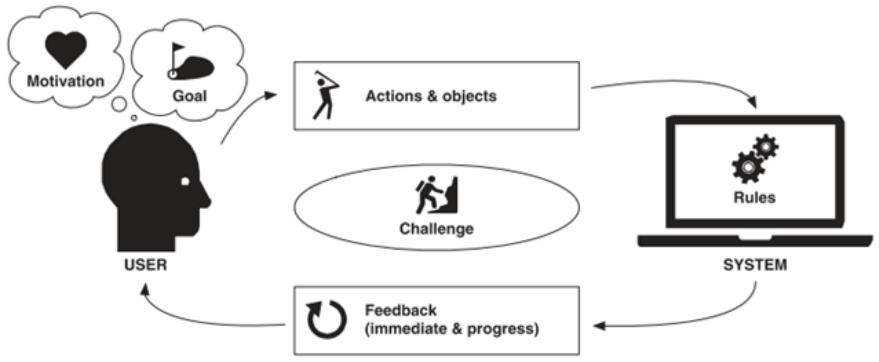


Figure 2.8: Deterding's skill atoms [31].

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

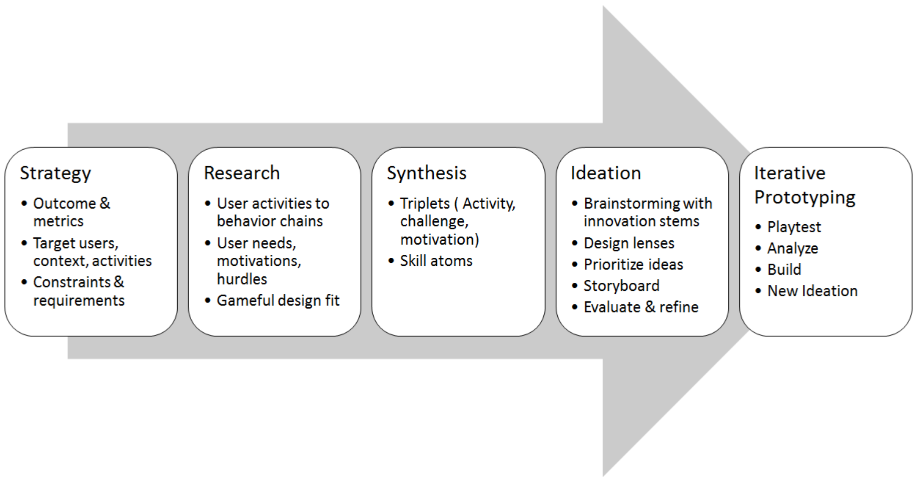

Strategy
• Outcome & metrics
• Target users, context, activities
• Constraints & requirements
Research
• User activities to behavior chains
• User needs, motivations, hurdles
• Gameful design fit
Synthesis
• Triplets (Activity, challenge, motivation)
• Skill atoms
Ideation
• Brainstorming with innovation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

Figure 2.9: Deterding's gameful design steps [31].

2.3 Software Modelling

While pedagogy provide methods for delivering learning contents and ludology contributes on improving the motivation in a learning process, software modelling is the actual learning topic that this research is concerned with. In this section, we describe and discuss what software modelling is, tools to perform software modelling, existing principles and practices in software modelling teaching, abstraction in software modelling and related studies of gamified software modelling.

2.3.1 What is Software Modelling?

Software modelling is a process of creating models of software. The process has a crucial role in Model Driven Engineering (MDE) methodology since MDE harnesses greatly the strength of modelling to improve engineering processes [41].

The process of modelling itself starts from a real world objects–a concrete experience–and then moves up to higher levels of abstraction [42, 4], which also known as a process of abstraction. Typical modelling architectures have four layers. The first layer is real world object, the second layer is model, the third one is metamodel, and the fourth one is metametamodel [41]. Models are abstractions of real-world objects. Models can also be abstracted in the form of metamodels, models that describe models. Likewise, the metamodels can be abstracted further in the form of metametamodels, models that define metamodels. In practice, metemetamodels can be defined in terms of themselves, in order to remain within 4 levels of abstraction, rather then creating infinite levels of modelling. This process of abstraction is depicted in Figure 2.10.

To create real working systems, models have to be transformed into software. We call this whole process as concretisation, a process that has a reverse direction to abstraction. Figure 2.11 illustrates a process of conretisation of a metamodel

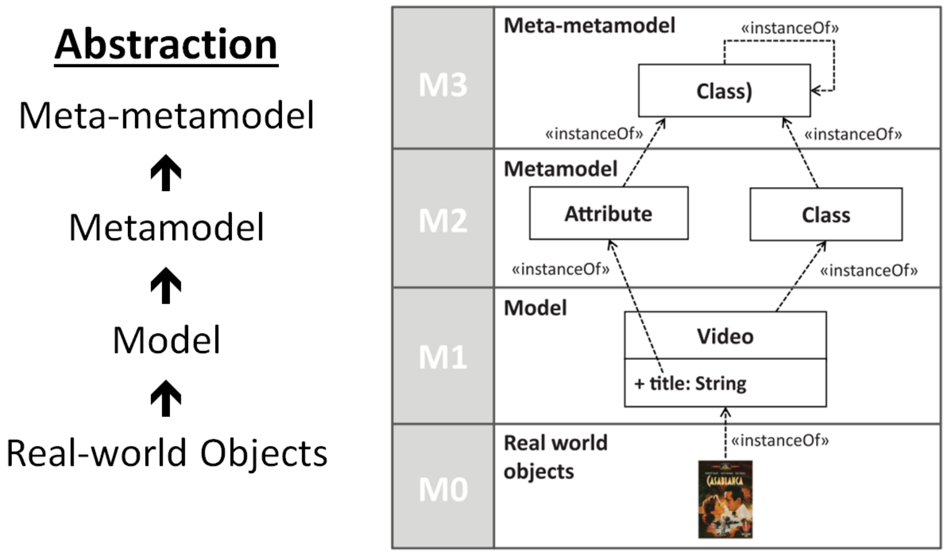


Figure 2.10: Abstraction in Model-driven Engineering [41].

of a flowchart model applied to waking-up process. The metamodel constitute the definition of a language that defines the model since it has rules to describe the model represented by the language [41]. So, every model created should conform to its metamodel. The metamodel is more general than it is models since it could be applied to different model domain. In Figure 2.11, the metamodel is instantiated to model the process of waking up. Through model transformation and instance generation, the model is taken as an input to generate a working system, in this example, a simple algorithm for waking up and turning off an alarm.

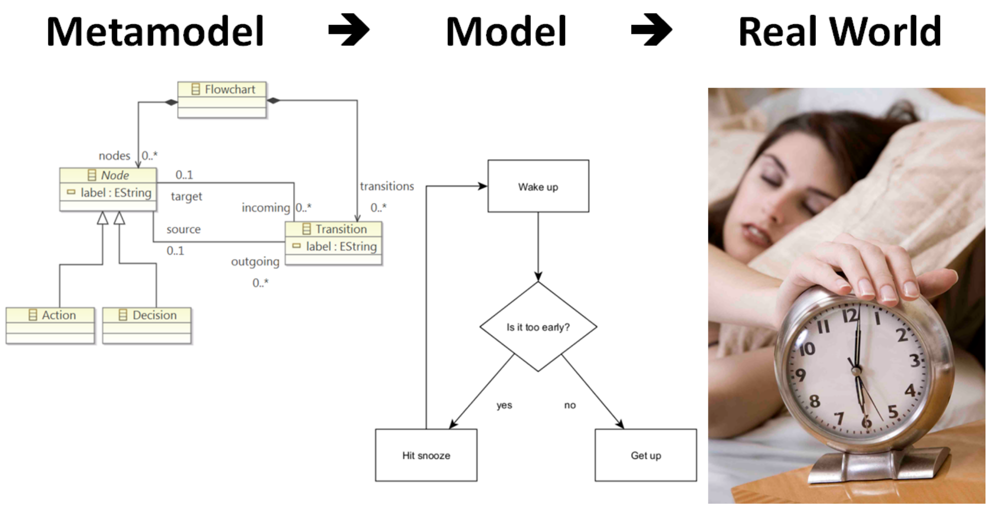


Figure 2.11: Instantiation in Model-driven Engineering.

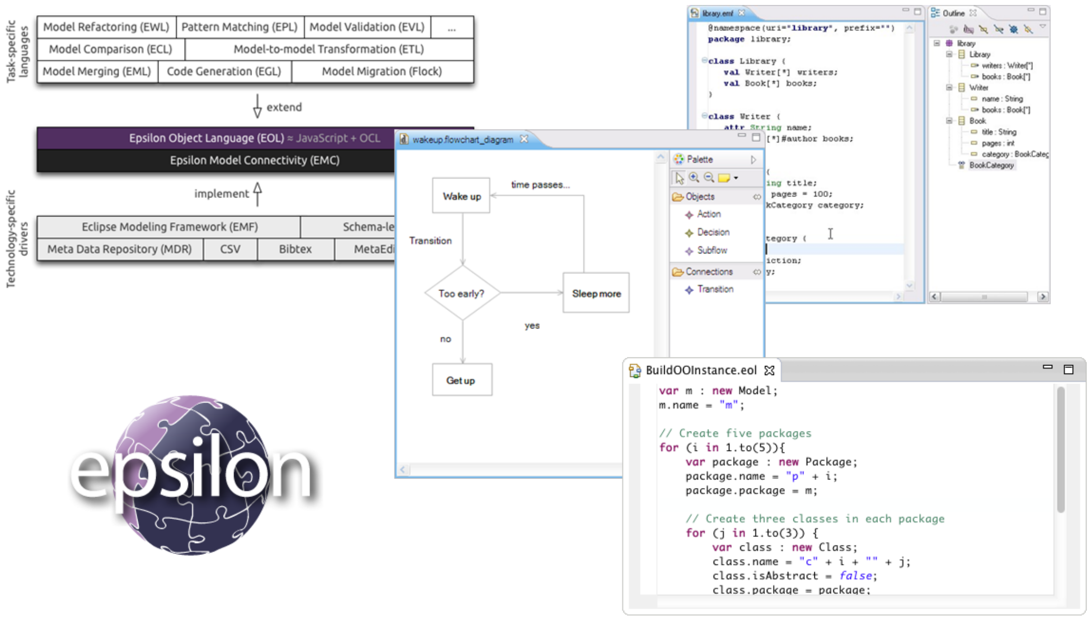
2.3.2 Software Modelling Tools

The use of tools for software modelling is encouraged since with them we could easily manipulate models, create models that at least conform to their metamodels, and some support model transformation [41]. There are many tools available for software modelling, ranging from general-purpose drawing tools like Dia[1] to language-specific modelling tools like Papyrus[2] and from simple modelling tools for learning [43] to enterprise-scale modelling suites [3].

For the purpose of learning, Dranidis et al. and Akayama et al. suggested developing simplified software modelling learning tools, since, for beginners, modelling using professional tools can be overwhelming [43, 44]. Once learners master the fundamentals of software modelling, they can move to more advanced topics, such as metamodelling and model transformation. One tool that supports advance model operations is Epsilon. Epsilon is a family of languages and tools for model management [45]. The model management itself consists of several processes. They are code generation, model-to-model transformation as well as model validation, com-

[1]http://dia-installer.de/
[2]https://eclipse.org/papyrus/
[3]https://www.visual-paradigm.com/features/

parison, migration and refactoring. Moreover, Epsilon also supports Eclipse Modeling Framework (EMF) and other model types, such Meta Data Repository (MDR), CSV, Bibtex, etc. (Figure 2.12). Epsilon also has an extension called EuGENia which are dedicated to produce graphical modelling editors automatically[46].



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

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

class Writer {
   attr String name;
   [*]#author books;

{
ing title;
 pages = 100;
kCategory category;

tegory {

iction;
y;
```

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

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

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

Figure 2.12: Epsilon is a family of languages and tools for model management [45].

2.3.3 Abstraction in Software Modelling Learning

Following the last discussion, modelling and meta-modelling, and additionally model management, are the potential concepts that will be taught in the game. All the three concepts require adequate abstraction skill, a skill that has a fundamental role in Computer Science and Software Engineering [4, 5, 6]. There are some available strategies in teaching abstraction. One of them are the Familiarity, Similarity, Reification, and Application steps proposed by White and Mitchelmore in the math education [42]. They argued that abstraction should be developed from empirical experience, move up to abstraction, and concrete the abstraction through the application. Similarly, based on their experience teaching UML, Engels et al. [4] approached abstraction through modelling according to this steps: real-world objects, object diagrams, class diagrams. The real-world objects can be described by videos or may be substituted by textual explanation, animations, or pictures. Hazzan [6] proposed three methods in teaching abstraction. First, illustrate. Lecturer uses abstraction-related words or statements in teaching. Second, reflect. For example, lecturer and students question the impact of using a certain level of abstraction or not using abstraction at all. Third, practice. Students must practice and reflect on what he do.

There are two approaches to learning software modelling which a learner may perform them simultaneously: rational approach and intuitive approach [47]. Usually at the beginning of the modelling process, one uses his existing knowledge and logical reasoning to understand and develop software models and sometimes preceded with empirical activities. However, there are situations in software modelling when his existing knowledge and methodology-based, logical reasoning are not enough to make him understand or to develop a software model. Thus, make him relies on his intuition or both approaches interplay in dealing with complex, difficult modelling. In line with Bobkowska's statement, in building a model on the correct level of abstraction, one needs a certain intuition and skill which cannot be gain through lectures but have to be experienced through exercises [4], which is also encouraged by experiential learning [25].

2.3.4 Software Modelling Teaching

We have investigated related literature regarding the teaching and learning of software modelling. From the literature, we have identified lessons and categorised them into three groups: contents, teaching or learning practices, and tool design. We will take into account the lessons learnt from the previous work during the design process of our gamified software modelling learning.

Contents

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

1. Software modelling definition [48].
2. Semantics, syntaxes, notations. Teach modelling foundations, but focus more on semantics, not only syntax [48]. Improve use of language, not only vocabulary [3].
3. Modelling, metamodelling, and model transformation. Focus on the core, important concepts: modelling, meta-modelling, and model transformation. Teach modelling that comprises the following topics: formal and informal models, partial and complete models, the distinction between models and programs [3]. Metamodelling is given a larger portion than modelling since a model might conform to more than one metamodel [49].
4. Software modelling is engineering. Teach the engineering aspects of software modelling: understanding a domain, planning and resourcing, documentation, quality, formality, validity, optimisation [50].
5. Contexts and practices/applications in various domains [50]. Teach the application of software modelling in various domains, success stories, code generations, model discovery, and model-driven interoperability. Make the model executable. Still, the power of execution makes it much easier to understand the model [3]. Convey the practical applications of modelling [48].

Teaching and Learning Practices

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

1. Modelling is the process to think abstractly about systems. Therefore, we teach modelling to make students understand the value of abstraction [3]. Successful application of Model-Driven Software Development requires skills in abstract modelling [2].

2. Teach with prerequisites [50]. The student should have a good programming background [3] or has to know a little about OOP [44].

3. Encourage students to produce "good" models and measure the "quality". One way is to use tools [44].

4. Learn modelling as early as possible [44], [48]. Teach modelling together with programming [48]. Modelling should be developed alongside programming [3].

5. Problem-solving first, modelling language specification and modelling tools get in the way [50].

6. Provide solutions, not answers [50].

7. Teach modelling language broadly, not deeply [50], and throughout [48]). Students need to experience the whole cycle of modelling in a software engineering project, so they learn to decide which development process is more appropriate than the others [44].

8. Refer to other disciplines or other aspects related to software modelling [50].

9. Even though code generation is essential to understand modelling [51], teach other applications (benefits) of software modelling at first. Code generation comes later [50].

10. Be careful when using analogies and physical decomposition since they might not reflect the complexity of the system; one component might have a cross-cutting effect to other layers of the system [50].

11. It is good to teach modelling with a standard language, such as UML [3], but teach modelling and meta-modelling using other modelling languages as well, not just UML. Software modelling is not a UML modelling course [50]. A significant number of successful Model-driven Software Development companies build their own modelling languages and generators, suggesting a re-orientation of education away from UML notation to fundamental modelling principles [2].

12. Choose a playful domain or fun problems, not serious domain [50].

Tool Design

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

1. Learning modelling tools is not trivial [50].
2. Build knowledge and skills incrementally [52].
3. Use papers, tools designed for pedagogy, or use mainstream tools [44].
4. The usage of modelling tools is not important in the beginning, but later when the modelling task becomes larger [3].
5. A good experience with the tool, has a positive inuence on the students view of UML and modelling in general [51].
6. Support for tool usage greatly affects student satisfaction. Provide Tool Expert and carefully design the instructions if you are using Industry Modelling Tool [51]. For example, Papyrus is suitable for a classroom environment if given the right level of support [51].
7. Give positive reinforcement to learners [52].
8. The tools offer maximum opportunities for learning [52].
9. The tool convinces learners of the value of what they learn [52].
10. Focus on high usability [52].
11. The tool is inexpensive [52].

2.3.5 Gamified Software Modelling

Most of the gamification studies available are dominantly related to software engineering in a larger context or other aspects of software engineering, such as software implementation and project management, rather than software modelling in particular [13]. Studies that apply gamification specifically for software modelling include the works of Stikkolorum et al. [53], Ionita et al. [54], Groenewegen et al. [55], and Richardsen [56].

Puzzle Game to Teach Software Design Principles

Stikkolorum et al. [53] developed a game that is intended to teach software design principles, such as cohesion, coupling, information hiding, and modularity in object-oriented software design. They tried to look for a solution that provides a balance between coupling and cohesion by using the toolbox to draw classes, methods, attributes, and relationships between them. They applied game elements like puzzles, game levels, visual and audio feedback, the progress indicator, level unlocking, choice of path, multiple solutions, and scoring. For the pedagogical aspect, they mentioned Blooms taxonomy in their work. However, they did not explain how they integrated the taxonomy into their work. After using their game, users started to refer to concepts such as classes, methods, and associations instead of boxes, blocks, and lines, indicating a progress of learning. The challenges of their work were on creating the best solution since there are more than one solutions for a problem. To determine coupling, they used Coupling Between Classes (CBO). Cohesion was measured by comparing all items—attributes, methods, class name—in a class that have similar keywords. Information hiding and modularity was evaluated using general design patterns. They validated their design by conducting user test and utilised the 'think aloud' method, asking users to tell their thoughts while using the game.

Explorable Board-game to Understand and Validate Enterprise Architecture

Groenewegen et al. [55] applied gamification to improve stakeholders' understanding of their enterprise architecture models as well as to validate them. They employed board game-like technique—exploring model step by step, element by element according to the given rules—which can provide a player with a progressive user experience. Therefore, it can improve user understanding. The game proposed is more playable than before, more freedom to try and explore, and no explicit rewards were given. For the game elements, they utilise cards, explorable board-game, and rules. Regardless of their claim that users can understand the model better rather than by merely looking at the model so they can argue whether the model is valid or not based on their existing knowledge, they did not discuss the pedagogical aspect of their work. The challenge that they experienced during the implementation is translating modeller's and reader's implicit knowledge unto explicit knowledge in the model—a gap of knowledge between the modeller and the reader, which domain knowledge is required. Lack of domain knowledge will make the model less understandable, and the user cannot validate the model. For validation, they tested their work to seven respondents and then interviewing them.

Familiar Tangible Model to Model Information Security

In the domain of information security modelling, Ionita et al.[54] developed a socio-technical modelling language (TREsPASS) that maps information security-related concepts toward tangible representation. Mapping the socio-technical modelling language to the tangible model, a model that uses physical objects as representation of concepts, is the most challenging part of their work. They expected that the tangible representations would increase the familiarity and understandability of models, which would raise awareness, involvement, and learnability. For the game elements, they utilise familiar, tangible representations, such as Lego characters, a board-game metaphor, and rules. The paper discusses the pedagogical aspect of their design which is based on constructivism, cognitive load, and cognitive fit. Moreover, based on their experiment, they reported that experimental group performed better than the control group in learning, efficiency, correctness, and satisfaction.

Likewise, based on an interview with experts and professionals, the respondents argued that the tangible model might be useful for less technical domain experts and different types of stakeholders to be more participative and contributive in the early stages of architecture modelling.

Arranging UML Activity Diagrams to Control the Behavior of a Game

In the context of activity diagram learning, Richardsen [56] developed a game the behaviours of which is controlled through arranging a UML activity diagram. Throughout his research, he found a challenge that controlling game from activity diagram in Reactive Block Environment[4], an eclipse-based visual development environment for Java application, is difficult since Eclipse[5] is difficult for a first-time user. For validation, he conducted user testing with three users. The 'think-aloud' method was used for observation. After that, questionnaires were given, and an interview was conducted. He found out there was no significant difference between the traditional interactive tutorial and the game-like tutorial on their performances. However, the game-like tutorial was more engaging. For the validation of his work, the author did not mention any explicit pedagogical aspects.

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

2.4 Research Methodology

A design framework for software modelling learning game will be derived deductively mainly from existing related theories and research works that come from the fields of education, games, and software modelling. The resulting framework will be tested in several case studies for validation, and the test results will be inputted to revise the framework. Since the outputs of the research will be designed artefacts, Design Science Research Methodology [57] is selected as the research methods as it provides a comprehensive conceptual framework and activity guidelines for understanding,

[4]http://reference.bitreactive.com/

[5]http://www.eclipse.org/home/index.php

developing, executing, and evaluating design artefacts. In the end of this section, we also discuss briefly our evaluation plan to evaluate our research findings.

2.4.1 Design Science Research Methodology

We will employ Design Science Research Methodology [57] as the methodology to carry out the research. DSRM is selected since it provides a comprehensive conceptual framework that consists of activity guidelines for understanding, developing, executing, and evaluating design artefacts (Figure 2.13). Oher 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.

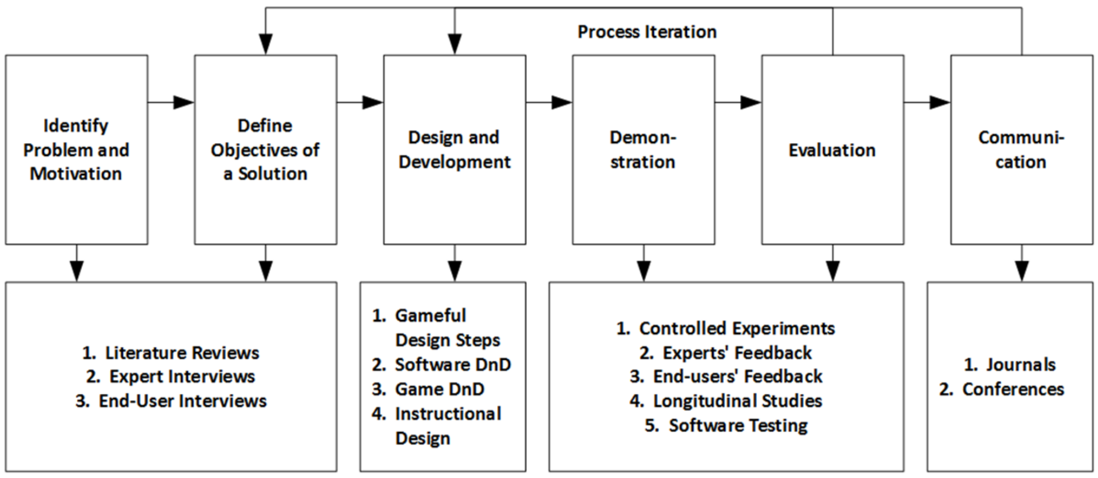


Figure 2.13: Design Science Research Methodology. Adapted from Peffer et al. [57].

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

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

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

Define Objectives of a Solution. Based on the identified problems and motivations, we have defined our solution that is embedding gamification in the process of software modelling learning, which is expressed by the use of an artefact specifically designed with gamification and pedagogy in mind. The research objectives and outcomes are defined in section 3.

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

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

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

2.4.2 Evaluation

We wish to evaluate (1) the effectiveness of the modelling games discussed above and (2) the productivity and maintainability benefits of the modelling game design framework. For the effectiveness evaluation, controlled experiments will be used. The participants, software modelling students, will be divided into two groups, a control group and an experimental group. The control group will learn software modelling using traditional methods while the experimental group will learn with support from the games. Then, their performance of the two groups will be measured by their ability to solve a set of related modelling problems.

For the evaluation of the modelling game design framework, the participants will be software modelling tutors; they will be divided into two groups, one that will develop games *with* the framework and one *without* the frameworks (i.e. using existing web technologies). They will be asked to elaborate their games into their teaching instructions and use them in their teaching. The comparison will be on their productivity and the maintainability of their games. To evaluate the generality of the results of both evaluation processes, conducting experiments in different years and countries is also considered.

Additionally, surveying with questionnaires or interviews might be conducted to investigate the underlying variables or processes. Structural equation modelling

[58] is also an option if measuring the effects of the identified underlying variables is required. An alternative method for understanding of the underlying variables and processes is through investigating the games' event logs using data mining or machine learning techniques.

2.4.3 Research Data Management

During the evaluation phase of this project, we plan to collect data from our respondents—part of research that is sensitive to privacy issues. Thus, as part of good research practices at the University of York[6], Research Data Management (RDM) is obligatory to ensure our research protects the privacy of our respondents, conforms to the law, and respects research ethics, as well as to demonstrate research excellence and integrity. Therefore, we plan to design a Data Management Plan (DMP) that refers to the Digital Curation Centre's outlines, which means the plan should address the following six main points: (1) Data Types, Formats, Standards, and Capture Methods. (2) Ethics and Intellectual Properties. (3) Access, Data Sharing, and Reuse. (4) Short-Term Storage and Data Management. (5) Deposit and Long-Term Preservation. (6) Outsourcing [59]. To support our work, we will use the DMP template[7] provided by the University of York for postgraduate research projects.

[6]https://www.york.ac.uk/library/info-for/researchers/data/
[7]https://www.york.ac.uk/library/info-for/researchers/data/

Chapter 3

Proposal

Grounded on the literature review in Chapter 2, in this this section, we present our research questions as well as our research aim and objectives to answer the questions. We also present the possible research outputs that will be produced during our research.

3.1 Research Questions

The main research question proposed by this research is ”Can gamification improve software modelling learning?” To answer this research question, following sub research questions need to be investigated:

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

3.2 Objectives

The main aim of this research is to assess to what extent gamification can improve software modelling learning. Therefore, we will investigate and develop a software modelling game design framework that systematically and semi-automatically drives gamification design to produce software modelling learning games. More precisely, this research aims to meet the following research objectives that are derived from the main research aim:

1. Perform a literature review to identify research problems, questions, and objectives.
2. Develop a conceptual framework of how to design gamification of software modelling learning based on the literature review and the survey. The framework will be iteratively updated according to the results obtained from controlled experiments.
3. Develop a software artefact as the instantiation of the conceptual framework to facilitate the gamification of software modelling learning. The product of the gamification will be tested to respondents for evaluation and to obtain feedbacks for iterative improvement.
4. Perform controlled experiments to measure the significance of the software modelling learning game in improving learning performance compared to traditional method, didactic learning without the support of the game.
5. Perform controlled experiments to measure the productivity and maintainability of a software modelling learning design framework in supporting tutors design and develop gamified software modelling learning.

3.3 Research Outputs

The potential research outputs of this research are:

1. Artefact. The software/application of gamification of software modelling learning.
2. Modelling Artefact. A tool for modelling the application of the gamification of software modelling learning.
3. Significance. Controlled experiments, learning outcome comparison between the gamified version and the traditional one.
4. Software Modelling Design Framework. Conceptual and software framework to perform gamification of software modelling learning.
5. Understanding. A model that explains how gamification of software modelling learning works. This could be achieved through Learning and Game Analytics and Structural Equation Modelling studies.
6. Case Study. Report the application of theories, models, and methods used in this research.

Chapter 4

Preliminary Results

In this section, we present our preliminary results. First, we present the result of our preliminary survey regarding the needs, motivation, and challenges of software modelling learning from the perspective of Model-Driven Engineering module's students. Next, we summarise our literature review regarding best practices in software modelling teaching into lists of design requirements. After that, we elaborate the potential contribution of the discussed learning models to the design of the software modelling learning game. In the end, we present the preliminary design of the game as well as its design framework.

4.1 Preliminary Survey

In this research, we have conducted our preliminary survey to identify learners' needs, motivations, and hurdles according to the Research phase of the Deterding's Gameful Design Framework. The activity 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 C.

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 takeaways from the preliminary interview regarding students' needs, motivations, and challenges, that can be implemented into the design of a gamified software modelling learning. *First*, the need of the students to learn MDE is to gain new 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 [25]. 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 [31].

The extraneous, unrelated activities were identified by asking question 8 and 9, which aimed at determining the uninteresting challenges. Most of the complaints 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 [51].

Table 4.1: Requirements derived from the preliminary survey (section 4.1).

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).
Un-interesting 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.

4.2 Design Requirements

Throughout the literature review, we have gathered requirements, which are the key points pointed out by MDE experts how we should teach MDE. Table 4.2 is a list of requirements summarised from the literature review in section 2.3.4. 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 software modelling learning game meets the current best teaching practices.

We also derive other requirements from our preliminary survey in section 4.1. These requirements (Table 4.1) 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 gamified software modelling learning. From our requirement identification, we found out that the items in the Contents category (Table 4.2) agrees with the item in the Needs category (Table 4.1). The finding suggests an agreement about the learning contents that should be delivered to students in learning MDE.

Table 4.2: Requirements derived from the literature review (section 2.3.4).

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

4.3 Elaborating Design and Learning Models

In section 2.1, we proposed several existing learning models that we will apply in the design process of our software modelling learning game. 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 game are depicted in in Figure 4.1 and Figure 4.2.

We decided to implement challenge as the fundamental game element that exists in the design of our game since it is one of the key features that exist in every game. The challenge is a crucial game element since it stimulates and provokes a player to engage with a game. 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 4.1). Bloom's taxonomy also act as an inventory of activities that provide us many options of activities that could give variability in our design.

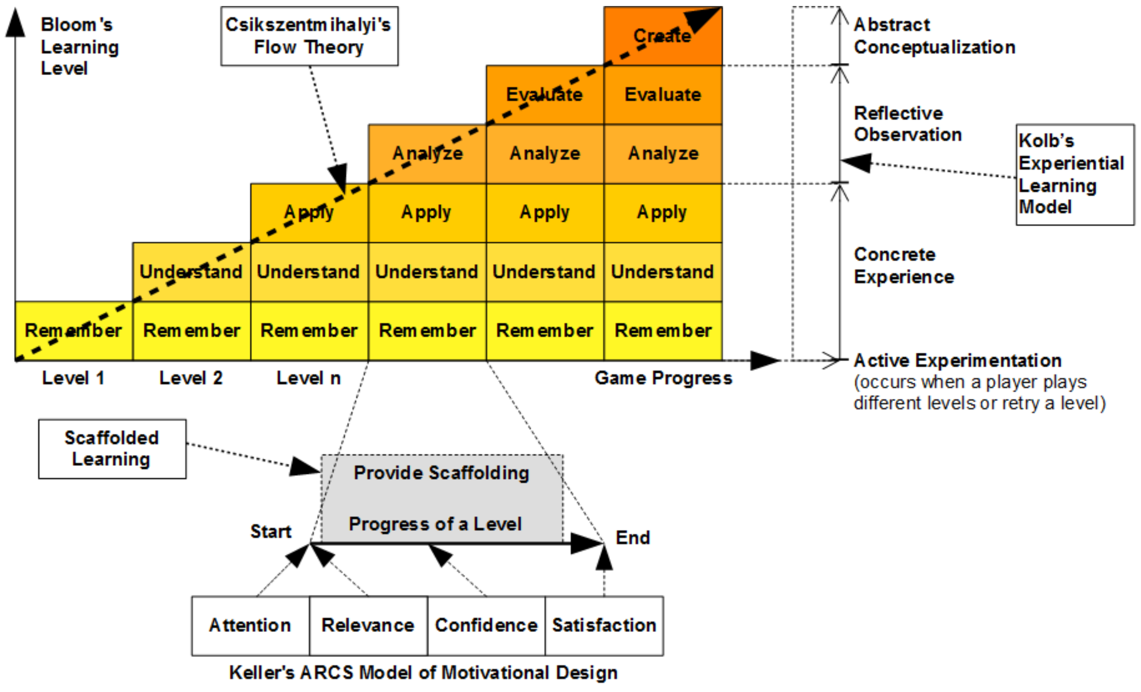


Figure 4.1: Elaborating learning models' contribution to the design of the gamified modeling learning

.

While learners are progressing in the game, 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

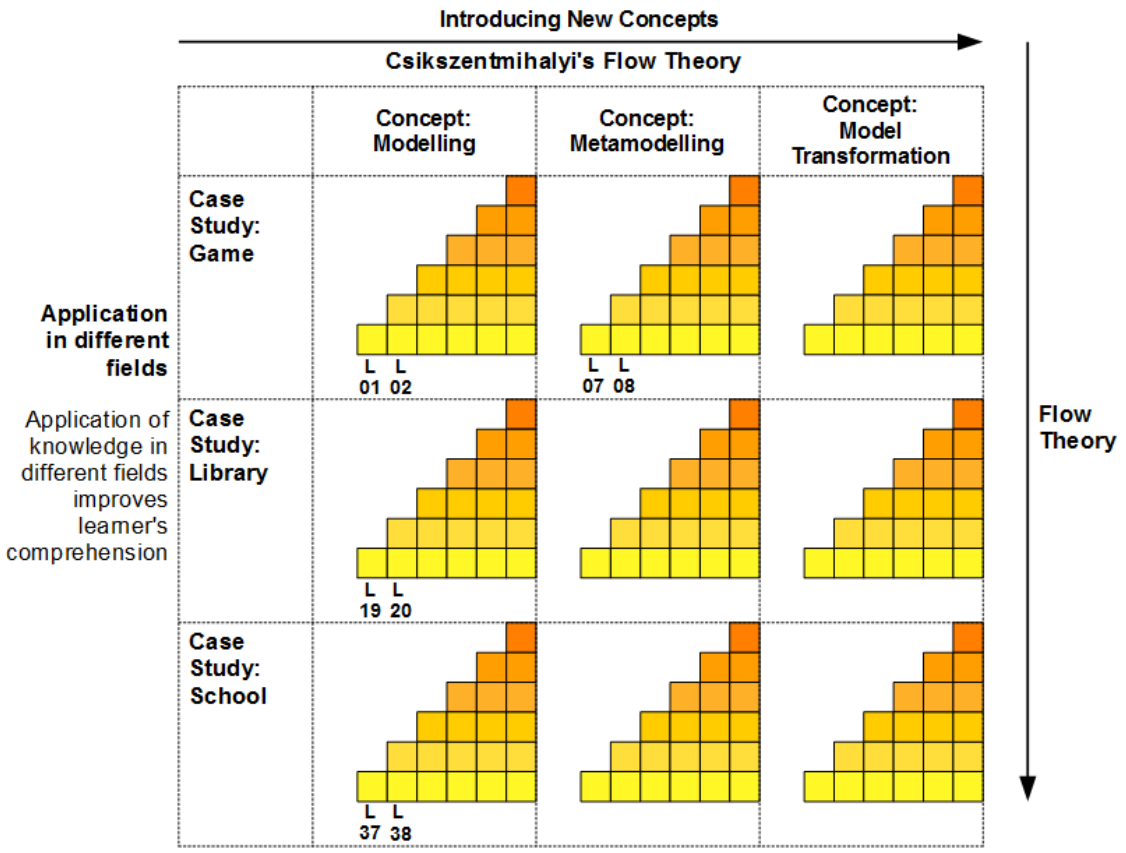

Introducing New Concepts
Csikszentmihalyi's Flow Theory
Concept: Modelling
Concept: Metamodelling
Concept: Model Transformation
Application in different fields
Application of knowledge in different fields improves learner's comprehension
Case Study: Game
Case Study: Library
Case Study: School
L 01
L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

Figure 4.2: Elaborating learning models' contribution to the design of the gamified modeling learning

.

(Figure 4.1). To control degree of difficulty, there are three ways we identified so far related to pedagogical approach: a combination of Bloom's activities, the introduction of new 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 4.1. Combining these three dimensions—types of activities, concepts, and domains—could give us a variety of levels with different degrees of difficulties.

Motivation is an important aspect in the success of learning, and we use Keller's ARCS motivational model to address this aspect [28]. We assume that one course of a level is similar to one course of a class meeting, from start to end and between then there are learning activities (Figure 4.1). The model provides us in each of it is components—attention, relevance, confidence, satisfaction—a set of predefined techniques to maintain learners' motivation. We could apply this technique also 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 will also apply scaffolding [17, 18] to support learners coping challenges (Figure 4.1). Throughout finishing a level, scaffolding could be provided in several ways: reducing extensive modelling activities into smaller activity constituents, removing irrelevant activities, providing an almost complete model so they can work on the most relevant activities rather than build the model from scratch, providing help and documentation, and giving some clues of the solutions when they get stuck. This support will be reduced as players progress to maintain the balance between their increasing competence and difficulty.

We also consider to apply Kolb's experiential learning model, which is a model that agrees knowledge is constructed through experience and based its model on constructivism [25]. We select Kolb's model since we perceive that playing a level in a game 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 4.3: Requirements derived from learning models (section 4.3).

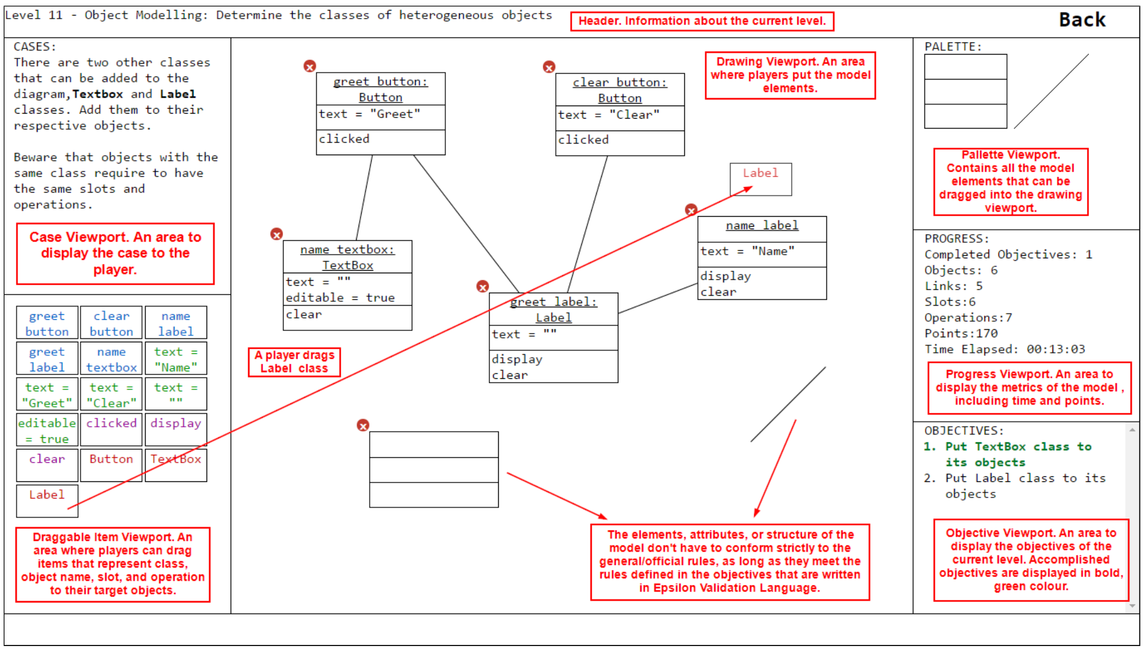
Category	Code	Requirements from Learning Models
Learning Models	RM01	Design satisfies Bloom's taxonomy.
	RM02	Design suffices Kolb's experiential learning model.
	RM03	Design meets Keller's ARCS motivational model.
	RM04	Design fulfils scaffolded learning.
	RM05	Design complies with the theory of Flow.

Learning activities in Bloom's taxonomy also correspond to the steps in Kolb's learning cycle [60] and both have been applied together to design instructions in different fields [61, 62, 63]. 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 4.3) that will be used in the design and evaluation activities.

4.4 Game Design

Gamification has been successfully for a variety of purposes, but there is very little work on software modelling gamification. We wish to assess whether gamification is beneficial for learners of graphical software modelling languages. 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 ensure the generality of our gamification solution, we plan to support several modelling languages. For each modelling language, we envision the development of a dedicated game containing game elements that will be derived from the Gameful Design Framework [31]. Each game will mimic a graphical modelling tool, and at each level, it will require the learner to graphically construct or adapt a model to satisfy a set of requirements and constraints.



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

Header. Information about the current level.

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

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

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

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

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

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

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

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

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

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

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

Figure 4.3: The display of the generated game.

A game 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 game to help learners familiarise themselves with the control system and the flow of the game.

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

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

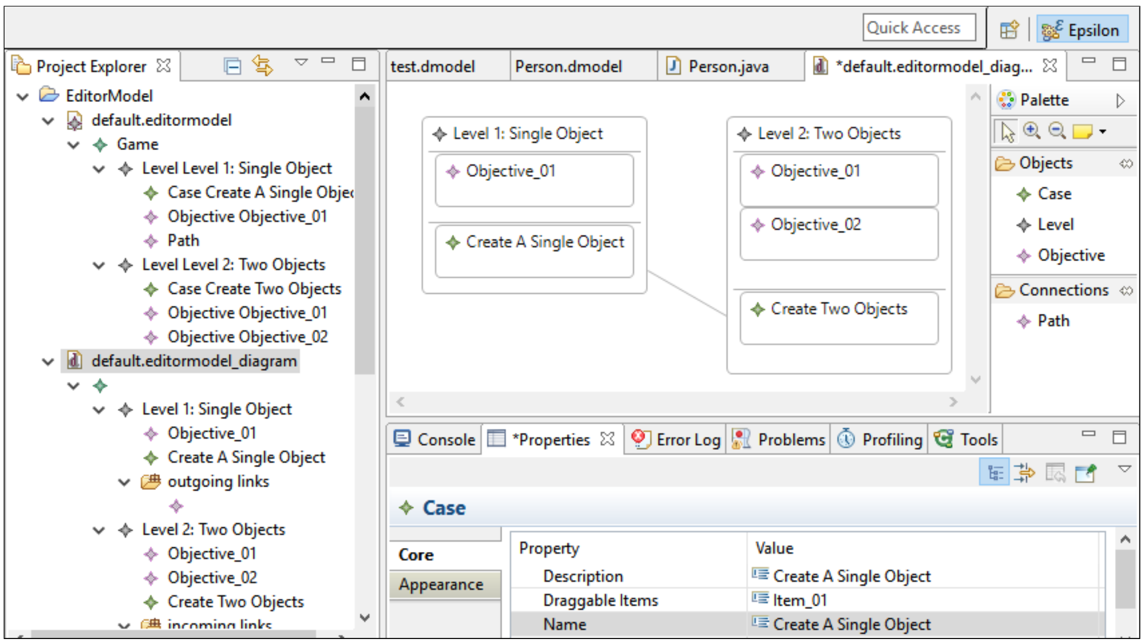
A real-world problem can be very complex and time-consuming to model. Thus, the extraneous activities that are not relevant to the core concepts that are being taught should be removed. As a result, learners will be more focused on the main concepts. Thus, game elements like bite-sized actions (e.g. drag and drop), limited choices (i.e. only limited items can be dragged), and microflows (i.e. put the right element to its right place) will be implemented to facilitate learners in performing the core activities. Likewise, fuzziness will also be used to provoke learners' creativity since most of the time there is no single correct model for the problem at hand. Attractive design will also be significant to motivate learners to interact with the game. Games should be able to give immediate, glanceable, and actionable feedback to keep learners on track and monitor their progress. Interesting and varied feedback should be designed to appeal to the learners' motives.

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

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

4.5 Modelling Game Design Framework

Instead of developing a software modelling game for each graphical modelling language manually, we plan to follow a model-based approach. We will use metamodel annotations, in the spirit of Eugenia [46], to define the graphical syntaxes of modelling languages and separate models to specify the game elements (levels, objectives, constraints, etc.) of each game. These models will be then consumed by a model-to-text transformation to produce fully-functional language-specific games. Therefore, the framework supports software modelling tutors in the design and customisation of the games at the high level of abstraction and so as to automatically build the game. So far we have implemented a metamodel for specifying game elements (flows, levels, challenges, and objectives) and a supporting Eclipse-based graphical editor (Fig. 4.4), and a prototype game (Fig. 4.3) for object diagrams.



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

Figure 4.4: Graphical editor for the game specification DSL.

Bibliography

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

[7] 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.

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

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

[15] J. Chen, "Flow in games (and everything else)," *Communications of the ACM*, vol. 50, no. 4, pp. 31–34, 2007.

[16] L.-F. Liao, "A flow theory perspective on learner motivation and behavior in distance education," *Distance Education*, vol. 27, no. 1, pp. 45–62, 2006.

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

[18] 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.

[19] G. Y.-M. Kao, C.-H. Chiang, and C.-T. Sun, "Designing an educational game with customized scaffolds for learning physics," in *Advanced Applied Informatics (IIAI-AAI), 2015 IIAI 4th International Congress on.* IEEE, 2015, pp. 303–306.

[20] F.-H. Tsai, C. Kinzer, K.-H. Hung, C.-L. A. Chen, and I.-Y. Hsu, "The importance and use of targeted content knowledge with scaffolding aid in educational simulation games," *Interactive Learning Environments*, vol. 21, no. 2, pp. 116–128, 2013.

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

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

[23] 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.

[24] C. G. von Wangenheim, R. Savi, and A. F. Borgatto, "Scrumiaan educational game for teaching scrum in computing courses," *Journal of Systems and Software*, vol. 86, no. 10, pp. 2675–2687, 2013.

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

[26] M. Bliemel and H. Ali-Hassan, “Game-based experiential learning in online management information systems classes using intel’s it manager 3,” *Journal of Information Systems Education*, vol. 25, no. 2, p. 117, 2014.

[27] L. Boctor, “Active-learning strategies: the use of a game to reinforce learning in nursing education. a case study,” *Nurse education in practice*, vol. 13, no. 2, pp. 96–100, 2013.

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

[29] W.-H. Huang, W.-Y. Huang, and J. Tschopp, “Sustaining iterative game playing processes in dgbl: The relationship between motivational processing and outcome processing,” *Computers & Education*, vol. 55, no. 2, pp. 789–797, 2010.

[30] L. Derbali and C. Frasson, “Players motivation and eeg waves patterns in a serious game environment,” in *International Conference on Intelligent Tutoring Systems.* Springer, 2010, pp. 297–299.

[31] 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.

[32] C. C. Abt, *Serious games.* University Press of America, 1987.

[33] K. Werbach, “(re) defining gamification: A process approach,” in *International Conference on Persuasive Technology.* Springer, 2014, pp. 266–272.

[34] A. R. Yohannis, Y. D. Prabowo, and A. Waworuntu, “Defining gamification: From lexical meaning and process viewpoint towards a gameful reality,” in *Information Technology Systems and Innovation (ICITSI), 2014 International Conference on.* IEEE, 2014, pp. 284–289.

[35] R. Garris, R. Ahlers, and J. E. Driskell, “Games, motivation, and learning: A research and practice model,” *Simulation & gaming*, vol. 33, no. 4, pp. 441–467, 2002.

[36] A. Yusoff, R. Crowder, L. Gilbert, and G. Wills, “A conceptual framework for serious games,” in *2009 Ninth IEEE International Conference on Advanced Learning Technologies.* IEEE, 2009, pp. 21–23.

[37] S. de Freitas and F. Liarokapis, “Serious games: a new paradigm for education?” in *Serious games and edutainment applications.* Springer, 2011, pp. 9–23.

[38] Y.-K. Chou, “Octalysis: Complete gamification framework,” *Yu-Kai Chou & Gamification*, 2013.

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

[40] N. B. Kumar, "A framework for designing gamification in the enterprise," *Infosys Labs Briefings*, vol. 11, no. 3, pp. 8–13, 2013.

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

[42] P. White and M. C. Mitchelmore, "Teaching for abstraction: A model," *Mathematical Thinking and Learning*, vol. 12, no. 3, pp. 205–226, 2010.

[43] D. Dranidis, I. Stamatopoulou, and M. Ntika, "Learning and practicing systems analysis and design with studentuml," in *Proceedings of the 7th Balkan Conference on Informatics Conference.* ACM, 2015, p. 41.

[44] 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.

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

[46] 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.

[47] A. E. Bobkowska, "Software modeling from the perspective of intuitive information processing," in *Proceedings of the 2014 Mulitmedia, Interaction, Design and Innovation International Conference on Multimedia, Interaction, Design and Innovation.* ACM, 2014, pp. 1–8.

[48] 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.

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

[50] 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.

[51] 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.

[52] 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.

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

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

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

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

[57] 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.

[58] J. F. Hair Jr, G. T. M. Hult, C. Ringle, and M. Sarstedt, *A primer on partial least squares structural equation modeling (PLS-SEM).* Sage Publications, 2016.

[59] S. Jones, *How to develop a data Management and sharing plan.* Digital Curation Centre, 2011.

[60] 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.

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

[62] 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.

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

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

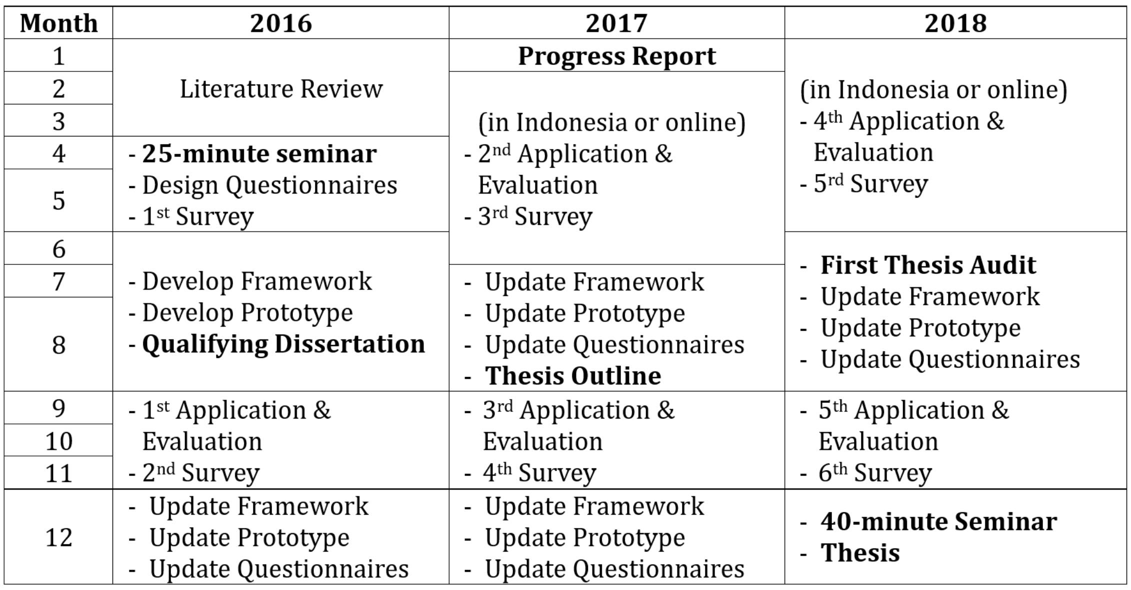
Appendix A

Research Plan

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

The whole process of abstraction and concretisation in MDE—from real world objects to metametamodels and vice versa—is the learning content that should be addressed in our gamification. We plan to address modelling in the 1st year and metamodelling in the 2nd year of our research. Model transformation will be addressed in the 3rd year. Appendix A shows the detailed research plan.

Table A.1: Research Timetable



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

Appendix B

Publications

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

1. A. Yohannis, Gamication of software modelling, in the ACM/IEEE 19th International Con- ference on Model Driven Engineering Languages and Systems (MODELS 2016) Doctoral Sym- posium. CEUR, 2016 [64].

Appendix C

Preliminary Survey Data

1. **If you think back to the time when you were just about to start the MODE module, what did you think you would find interesting in Model Driven Engineering?**
    - Learning what MDE actually is. Had never heard of it so was intrigued.
    - I thought I would be provided with a high level approach to designing and managing software/critical systems.
    - Learning some tools to create automodels.
    - Learning about ways of statically verifying that code conformed to a formal model, and using this to detect and 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 hurdles—the noninteresting challenges—that hindered or demotivated you in learning ModelDriven Engineering? Why?**

    - Learning the Eclipse Modelling Framework, MOF, etc. wasn't fun. The most fun part was learning and use Emfatic with EGL/EGX and EOL in general. 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 hurdles?**

    - 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 D

Application of Deterding's Gameful Design Steps

D.1 Strategy

D.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.

D.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

D.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.

D.2 Research

D.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

D.2.2 Identify User Needs, Motivations, Hurdles

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.

Hurdles

- 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.

D.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**

D.3 Synthesis

D.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 (+)

D.4 Ideation

D.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?

D.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.

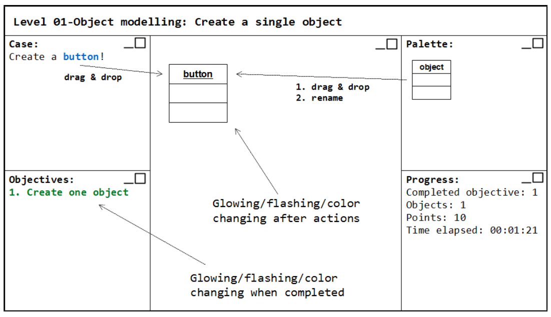
D.5 Iterative Prototyping

Prototyping is still on going work.

Appendix E

Storyboard: Object Diagram

- Objects
  - Create single object (Level 01)
  - Create two objects (Level 02)
  - Create multiple objects (Level 03)
- Links (Relationships)
  - Create single link (Level 04)
  - Create multiple links (Level 05)
- Slots (Attributes)
  - Determine a slot and its value (Level 06)
  - Determine slots and their values (Level 07)
- Operations/Methods
  - Determine operation (Level 08)
  - Determine multiple (Level 09) operations
- Class of Objects
  - Determine the class of homogeneous objects (Level 10)
  - Determine different classes of heterogeneous objects (Level 11)
- Case Studies
  - Reconstruct the model from the beginning (Level 12)
  - Apply the skills on different/similar problem (Level 13)


Level 01-Object modelling: Create a single object

Case:
Create a button!

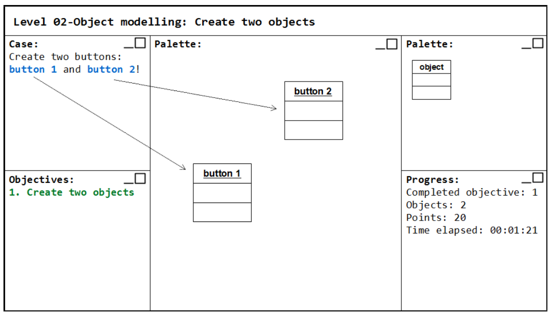
Palette:

Objectives:
1. Create one object

Progress:
Completed objective: 1
Objects: 1
Points: 10
Time elapsed: 00:01:21

drag & drop
button
1. drag & drop
2. rename
object
Glowing/flashing/color changing after actions
Glowing/flashing/color changing when completed

(a) Level 01


Level 02-Object modelling: Create two objects

Case:
Create two buttons:
button 1 and button 2!

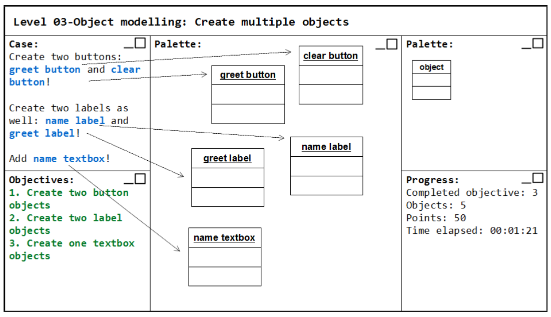
Palette:

Palette:

Objectives:
1. Create two objects

Progress:
Completed objective: 1
Objects: 2
Points: 20
Time elapsed: 00:01:21

(b) Level 02


Level 03-Object modelling: Create multiple objects

Case:
Create two buttons:
greet button and clear button!

Create two labels as well: name label and greet label!

Add name textbox!

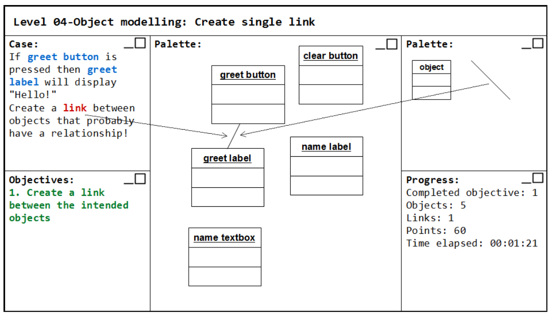
Palette:

Palette:

Objectives:
1. Create two button objects
2. Create two label objects
3. Create one textbox objects

Progress:
Completed objective: 3
Objects: 5
Points: 50
Time elapsed: 00:01:21

(c) Level 03


Level 04-Object modelling: Create single link

Case:
If greet button is pressed then greet label will display "Hello!"
Create a link between objects that probably have a relationship!

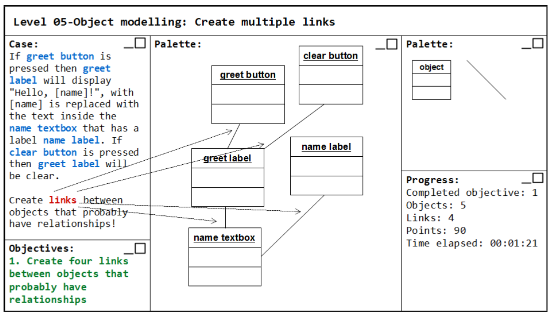
Palette:

Palette:

Objectives:
1. Create a link between the intended objects

Progress:
Completed objective: 1
Objects: 5
Links: 1
Points: 60
Time elapsed: 00:01:21

(d) Level 04


Level 05-Object modelling: Create multiple links

Case:
If greet button is pressed then greet label will display "Hello, [name]!", with [name] is replaced with the text inside the name textbox that has a label name label. If clear button is pressed then greet label will be clear.

Create links between objects that probably have relationships!

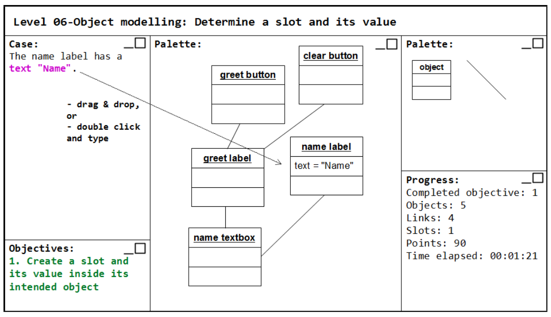
Palette:

Palette:

Objectives:
1. Create four links between objects that probably have relationships

Progress:
Completed objective: 1
Objects: 5
Links: 4
Points: 90
Time elapsed: 00:01:21

(e) Level 05


Level 06-Object modelling: Determine a slot and its value

Case:
The name label has a text "Name".

Palette:

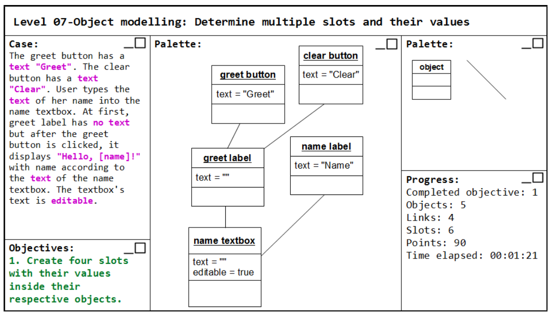
Palette:

Objectives:
1. Create a slot and its value inside its intended object

Progress:
Completed objective: 1
Objects: 5
Links: 4
Slots: 1
Points: 90
Time elapsed: 00:01:21

(f) Level 06

Figure E.1: Storyboard of Gamification of Object Diagram (part 1).

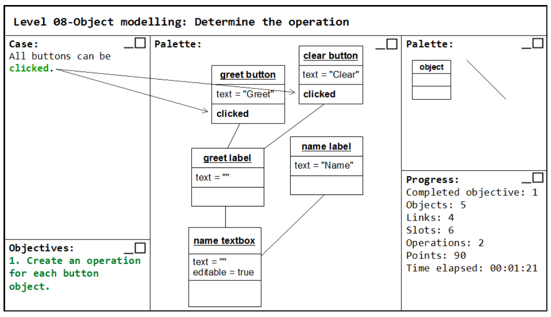

Level 07-Object modelling: Determine multiple slots and their values

Case:
The greet button has a text "Greet". The clear button has a text "Clear". User types the text of her name into the name textbox. At first, greet label has no text but after the greet button is clicked, it displays "Hello, [name]!" with name according to the text of the name textbox. The textbox's text is editable.

Objectives:
1. Create four slots with their values inside their respective objects.

Progress:
Completed objective: 1
Objects: 5
Links: 4
Slots: 6
Points: 90
Time elapsed: 00:01:21

(a) Level 07

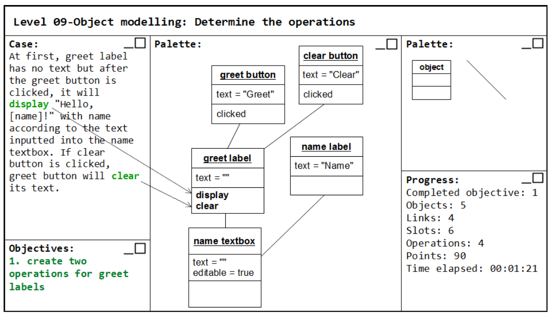

Level 08-Object modelling: Determine the operation

Case:
All buttons can be clicked.

Objectives:
1. Create an operation for each button object.

Progress:
Completed objective: 1
Objects: 5
Links: 4
Slots: 6
Operations: 2
Points: 90
Time elapsed: 00:01:21

(b) Level 08

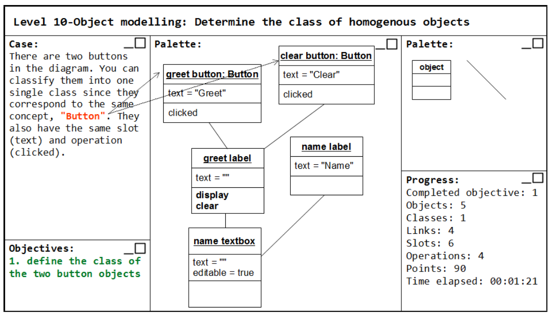

Level 09-Object modelling: Determine the operations

Case:
At first, greet label has no text but after the greet button is clicked, it will display "Hello, [name]!" with name according to the text inputted into the name textbox. If clear button is clicked, greet button will clear its text.

Objectives:
1. create two operations for greet labels

Progress:
Completed objective: 1
Objects: 5
Links: 4
Slots: 6
Operations: 4
Points: 90
Time elapsed: 00:01:21

(c) Level 09

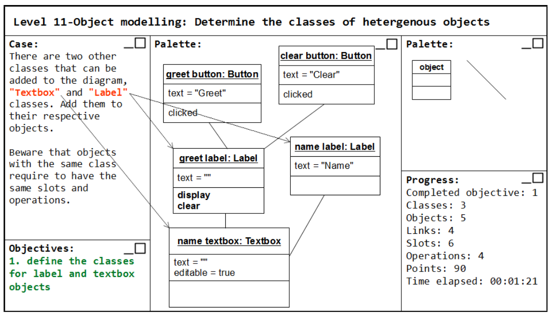

Level 10-Object modelling: Determine the class of homogenous objects

Case:
There are two buttons in the diagram. You can classify them into one single class since they correspond to the same concept, "Button". They also have the same slot (text) and operation (clicked).

Objectives:
1. define the class of the two button objects

Progress:
Completed objective: 1
Objects: 5
Classes: 1
Links: 4
Slots: 6
Operations: 4
Points: 90
Time elapsed: 00:01:21

(d) Level 10


Level 11-Object modelling: Determine the classes of hetergenous objects

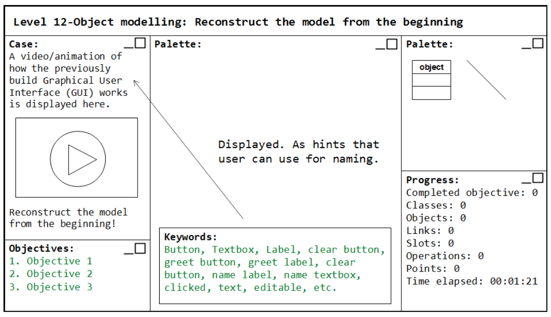
Case:
There are two other classes that can be added to the diagram, "Textbox" and "Label" classes. Add them to their respective objects.

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

Objectives:
1. define the classes for label and textbox objects

Progress:
Completed objective: 1
Classes: 3
Objects: 5
Links: 4
Slots: 6
Operations: 4
Points: 90
Time elapsed: 00:01:21

(e) Level 11


Level 12-Object modelling: Reconstruct the model from the beginning

Case:
A video/animation of how the previously build Graphical User Interface (GUI) works is displayed here.

Reconstruct the model from the beginning!

Objectives:
1. Objective 1
2. Objective 2
3. Objective 3

Displayed. As hints that user can use for naming.

Keywords:
Button, Textbox, Label, clear button, greet button, greet label, clear button, name label, name textbox, clicked, text, editable, etc.

Progress:
Completed objective: 0
Classes: 0
Objects: 0
Links: 0
Slots: 0
Operations: 0
Points: 0
Time elapsed: 00:01:21

(f) Level 12

Figure E.2: Storyboard of Gamification of Object Diagram (part 2).