Qualifying Dissertation

Software Modelling Learning Gamification

Alfa Ryano Yohannis
ary506@york.ac.uk

Supervisor:
Dimitris Kolovos
Fiona Polack

Department of Computer Science
University of York
United Kingdom

December 5, 2016

Abstract

In software engineering, software modelling plays a significant role. Nevertheless, learners often consider software modelling as a comparatively difficult subject since it requires them to have abstraction skills to master it. Meanwhile, gamification has been growing as a trend solution to improving learners' engagement. This study endeavours to harness gameful design to build gamification that supports learners advancing their modelling abilities. Our method to dealing with gameful design combines pedagogical design principles derived from several learning models and the Deterding's Gameful Design framework as an approach to gamification development. This research also employs Model-Driven Engineering best practices and uses the Design Science Research Methodology. This research aims to produce software modelling learning gamification (SMLG), and an SMLG framework to design and generate the SMLG's instances. We plan to perform controlled experiments to evaluate the effectiveness of the SMLG and the SMLG framework.

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. 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 [5] and Hazzan [6] argued that abstraction is the central theme or key skill for computing.

> "I believe that abstraction is a key skill for computing. It is essential during requirements engineering to elicit the critical aspects of the environment ... At design time ... Even at the implementation stage ..." — Kramer [5].

> " ... software is an intangible object, and hence, requires highly developed cognitive skills for coping with different levels of abstraction." — Hazzan [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 effort into software modelling learning design, developing a concrete and motivating presentation which can engage students and facilitate deep learning.

In recent years, the use of games and game elements for purposes other than leisure has drawn significant attention. Gamification [9] and Serious Games [10] (we use 'gamification' to refer to both concepts) have been proposed as solutions to motivational problems that emerge when users are required to engage in activities

that they perceive as boring, irrelevant, or difficult. Figure 1.1 shows the growth of "Gamification" on Google Trends[1] since February 2010.

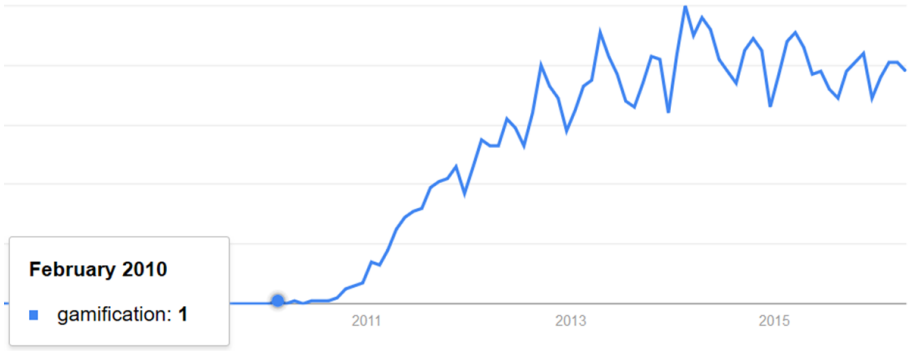


Figure 1.1: "Gamification" on Google Trends per 12 April 2016.

Real-world examples that show the success of the gamification are Duolingo[2] and Re-mission[3]. Duolingo is a gamified system of language learning. It embeds game elements, such as points, levels, and lives, to make language learning more fun. Re-mission is a third-person shooting game dedicated to young cancer patients and designed to teach and learn how to deal with cancer. The patients are invited to take part in an entertaining gameplay that will affect their specific behavioural and psychological outcomes producing effective cancer therapy.

Through a systematic review, Connolly et al. [11] studied the impact of computer games and serious games on engagement and learning in diverse fields. They reported the majority of the studies presented empirical evidence about the positive impact of computer games and serious games. Using the same type of method, Hamari et al. [12] found that according to the majority of the reviewed papers, gamification does generate benefits and positive effects. Specifically in the field of software engineering, Pedreira et al. [13] also performed a systematic review on the application of gamification in software engineering. Most existing studies focus on software development, project management, requirements, and other support areas, but none of them focuses on software modelling. They also found fewer studies reporting empirical evidence to support gamification research. They argued that existing studies in the field are quite new, thus more research effort is needed to investigate the impact of gamication in software engineering.

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

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

engagement of learners with the support of gamification is more durable, frequent, and active compared to learners that only use didactic approach. Also, the former perform better in knowledge and skill acquisition and application compared to the latter.

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

Instead of developing the software modelling games manually, we plan to follow a model-based approach. We will develop a design framework for the games, which will systematically and semi-automatically drive gamification design to produce software modelling learning gamification (SMLG). We call the framework the SMLG framework. 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. We also target the SMLG is suitable for higher-level undergraduate and postgraduate students with some of experience of software engineering.

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

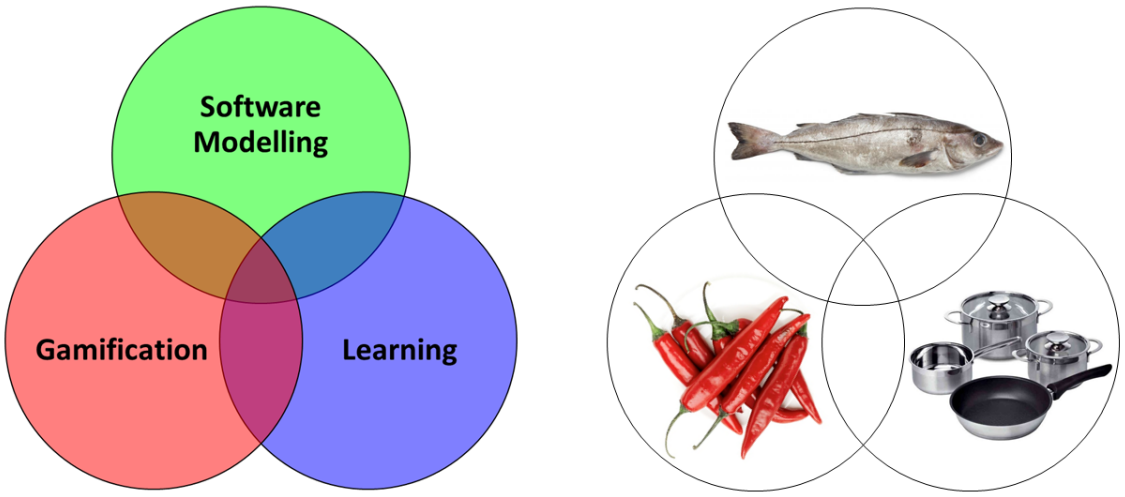


Figure 1.2: How to 'Cook' a Gameful Software Modelling Learning?

Chapter 2

Field Survey and Review

The research underpinning SMLG is multidisciplinary. Therefore, the literature review is organised by the contributing fields, including the subtopics resulting from the interaction between them. Three major contributing fields are pedagogy, ludology, and software modelling (Figure 2.1). They are discussed below, and a brief review of the research methodology concludes this chapter.

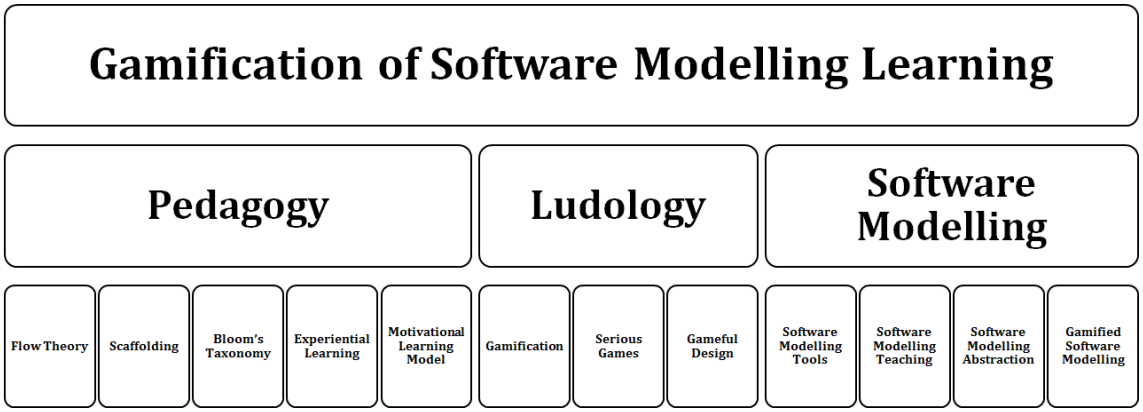


Figure 2.1: Contributing fields to the SMLG.

2.1 Pedagogy

Designing SMLG cannot be separated from the field of pedagogy since the core process supported by the SMLG 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 SMLG.

We have investigated literature in the field pedagogy regarding teaching and learning. The studies that we investigated are Flemming's VARK modality model [14], Flow theory [15], Revised Bloom's taxonomy [16], Gardner's multiple intelligence [17], Felder's learning styles [18], scaffolding [19, 20], experiential learning [21], self-determination theory [22], and Keller's motivational learning model [23]. However, we identified only five learning models that are relevant to our research

since they have characteristics that we considered will address certain design concerns of our SMLG. We excluded the self-determination theory from our research since one of its three components counts relatedness—one's need to get along with others—as one of the main motivations in learning, which we view is not relevant to this research. The rest are excluded since the studies focus on learning styles and preferences and why learning processes should accommodate them, which is not in line with our work that depends heavily on visual modality–extensive use of graphics for modelling. Therefore, we only discuss the selected five models in the following subsections.

2.1.1 Csikszentmihalyi's Flow Theory

Csikszentmihalyi [15] proposes 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. If the challenges are too difficult, they will arrive at a state of anxiety that demotivates them and then affects them to withdraw from the activity. On the other side, if the challenges are too easy and their competence is very advanced, they will be in the state of boredom which also makes it likely for him to withdraw from the activity (Figure 2.2).

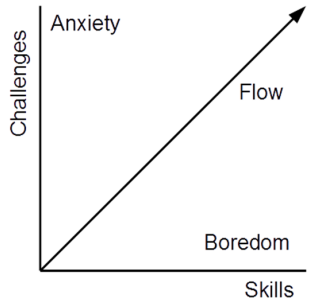


Figure 2.2: Csikszentmihalyi's Flow Theory [15].

Flow theory has been applied in many fields, such as education, training, and games to guide maintenance of engagement. 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 [24] comments that well-designed games should bring their players to the flow states, balanced states between challenges and skills, delivering happiness and pleasure. In the field of education, Liao [25] also applies the theory as a framework to study the emotional and cognitive responses to distance learning systems. He claims that flow theory, the cause and effect of the flow experience, also works when students use distance learning systems.

2.1.2 Zone of Proximal Development/Scaffolding

Zone of Proximal Development (ZPD) is proposed by Vygotsky in the context of adolescent development [20]. Wood et al. [19] use the term scaffolding to refer to the same concept in the context of learning. 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).

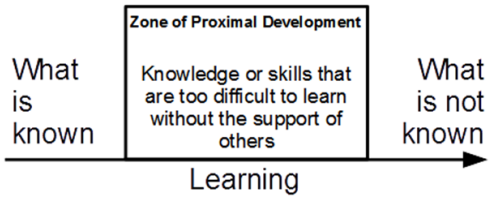


Figure 2.3: Zone of Proximal Development/Scaffolding [20],[19].

Kao et al. [26] study customised scaffolds of an educational game for learning physics, and Tsai et al. [27] investigate the use of scaffolding aids with importance and use of targeted content knowledge in educational simulation games. Both find proper scaffolding facilitates learning and helps learners to have better performance.

2.1.3 Revised Bloom's Taxonomy

Bloom's Taxonomy [16] 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 [28]. 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).

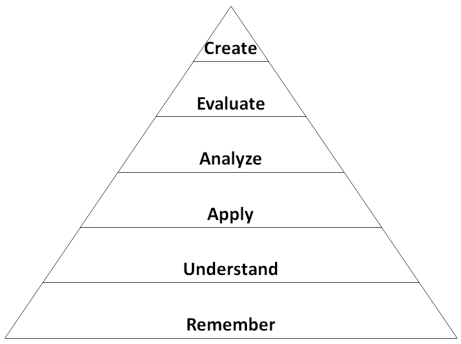


Figure 2.4: Revised Bloom's Taxonomy[16].

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 [29]. Von Vangenheim et al. [30] also developed an educational game for teaching SCRUM and applied activities in the taxonomy as a standard to assess the achievement of their respondents.

2.1.4 Kolb's Experiential Learning Model

Kolb's Experiential Learning Model [21] is a model of teaching and learning through experience and reflection on actions. It states that knowledge development is a product of experience or iterative search. The model is a cycle consisting of four steps: abstract conceptualisation, active experimentation, concrete experience, and reflective observation (Figure 2.5).

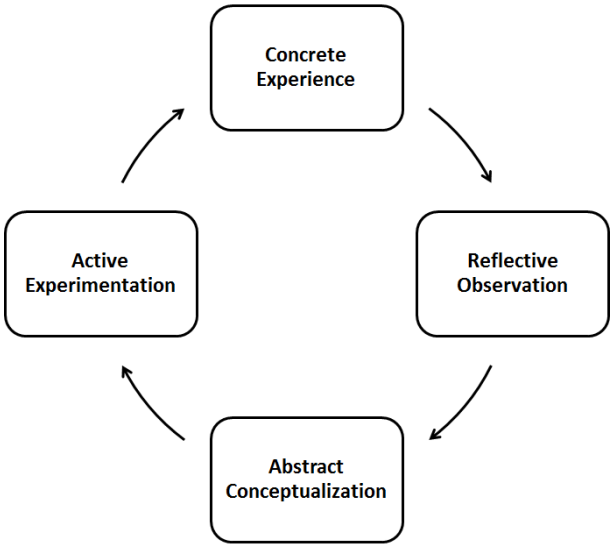


Figure 2.5: Kolb's Experiential Learning Model [21].

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 using the theory they were taught in the class. Through experimentation, they obtain concrete experience. The results might to some degree conform 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 proposes, since playing games demands learners to actively participate in their course, reflect on their achievements, strategies, and outcomes, and to plan their next actions carefully to improve their performance. As showcases, Bliemel and Hassan [31] use 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. Boctor [32] studies 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.

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 [23]. The steps are attention, relevance, confidence, and satisfaction (Figure 2.6). Attention means how to arouse learners to get their focus and awareness. Relevance uses 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. To realise it, 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	Objectives	Progress	Achievement
Surprise	Usefulness	Performance	Enjoyment
Curiousity	Instructions	Challenge	Rewards
Aesthetics	Motives	Sucess	Motivation
Questions	Contexts	Ability	Feedbacks

Figure 2.6: Keller's Motivational Model [23].

Huang et al. [33] conduct an experiment using the ARCS model to examine the motivational aspect when learners play digital game-based learning and find that motivational processing (attention, relevance, and confidence) need to be considered when designing digital game-based learning. Likewise, Derbali and Frasson [34] study players' motivation while playing a serious game. Using Keller's motivational model and electroencephalography to measure learners' motivation, they identify that learning using serious game significantly increases learner motivation.

2.2 Ludology

Ludology, the study of games, clearly is of significant relevance to our work, since it provides principles that guide the design of the gameful aspect of the SMLG. Concepts that are from Ludology, such as serious games, gamification, and gameful design framework [35] are discussed briefly in the following sections.

2.2.1 Serious Games and Gamification

Two terms related to games are used throughout this report: serious games and gamification. Abt [36] defines serious games "games that have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement", whereas Deterding et al. [9] define gamification as "the use of game design elements in non-game contexts". While both are used for purposes other than leisure, there is a 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.

Another confusion concerns on the word 'gamification'. The word 'gamification' could also be defined as a process [37]. The process adds game elements to an entity 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 [38]. In other words, the gamification process could transform an entity into a game, but the entity could still own its original characteristics.

To avoid 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. Similarly, 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. Therefore in this report, instead of using 'game' as the word that refers to the product of our gamification process, we use the word 'gamification' and specifically use 'SMLG' acronym to refer to software modelling learning gamification. We use the word 'game' only to refer to games that are commonly known.

2.2.2 Gameful Design Framework

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. Most of them are specific in a pedagogical context, but they only consider the use of full-fledged games, not game elements, in their research [39, 40, 41]. Others act as an inventory of problem-solution mappings where designers could choose one or more game mechanics that are appropriate to their problems [29, 42]. The rest are general procedural steps and raised from non-pedagogical contexts [43, 44, 45] and specifically intended for game design [46].

However, none of the game and gamification design-related frameworks provides design guides that are relevant, reflective, innovative, and generic altogether to produce a gameful design. Relevant means gamification ideas should arise from stakeholders' needs and contexts. Reflective means a deep thinking about the interconnection between game elements and core activities that are going to be gamified. Innovative suggest that a gameful design framework should not be limited to the adoption of mechanics from a fixed inventory, but it should be stimulating enough to produce new forms of mechanics that fit with the stakeholder-specific aspects.

Generic implies that the framework should be applicable in the domain of software modelling learning.

Deterding's Gameful Design framework [35] is a framework that could facilitate our purpose. 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 a collection of game elements (game elements 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 [35]. A design lens is a combination of an easy-to-remember name, a brief explanation of a design principle, and questions that help designers to view activities, motivation, and challenges from the perspective of game design principle that the lens represents. Deterding's Design Lenses [35] come with four categories of lenses: challenge, goal and motivation, action and object, and feedback. Each category comes with lenses to reflect on the activities, motivation, and challenges from game element perspective to generate innovative ideas of gamification (Figure 2.7).

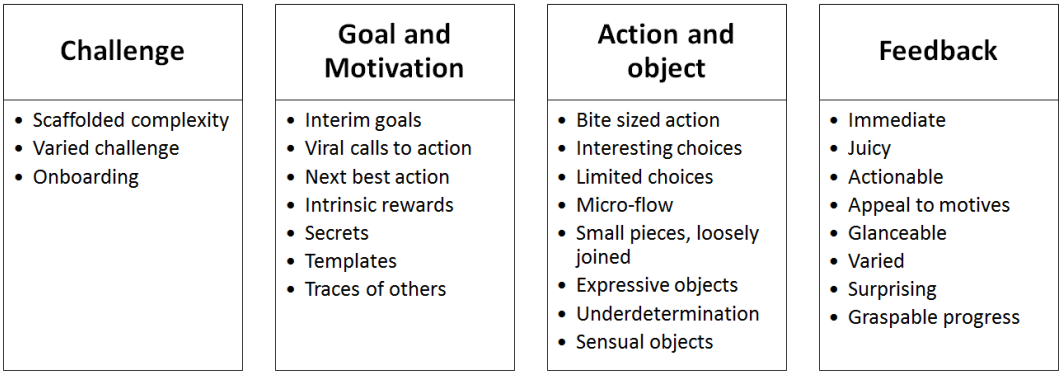


Figure 2.7: Deterding's design lenses [35].

A skill atom describes a feedback loop between a user and 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 then returns feedback to the user, which will be integrated into the user's mental model of the system. Through repeated action-feedback interaction, the user masters the intended skill.

The prescriptive guidelines of the Gameful Design framework comprise five phases: strategy, research, synthesis, ideation, and iterative prototyping. In the strategy phase, stakeholders define outcome metrics, target users, context, and activities as well as constraints and requirements. In the next phase, the research phase, user activity, behaviour chains, user needs, motivations, and challenges are identified and defined. After that, in the synthesis phase, activity-challenge-motivation triplets are defined, including defining skill atoms. Next, in the ideation phase, ideas are generated for the gamification design, brainstorming with innovation stems is

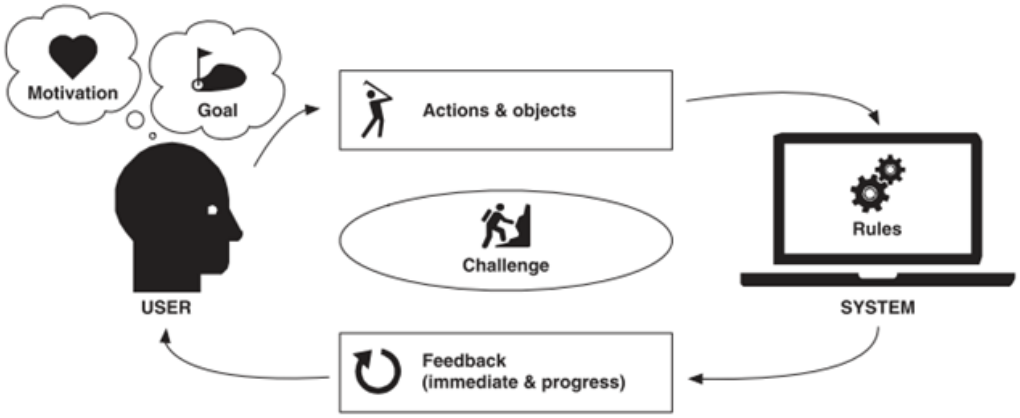


Figure 2.8: Deterding's skill atoms [35].

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.

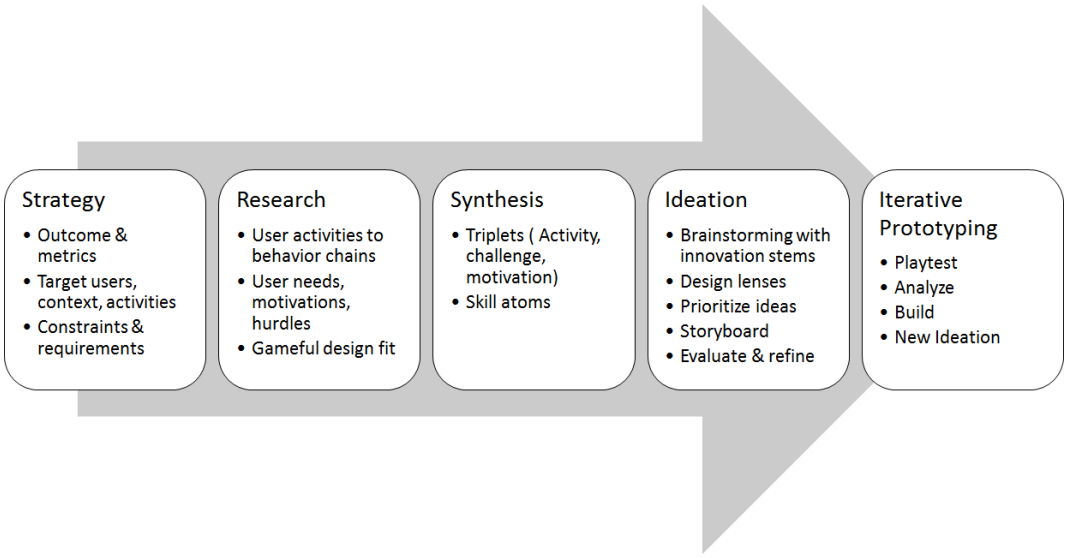


Figure 2.9: Deterding's gameful design steps [35].

2.3 Software Modelling

While pedagogy provides methods for delivering learning contents and ludology contributes to improving the motivation in a learning process, software modelling is the actual learning topic that this research is concerned with. In this section, we 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 creates abstract models of domains to explore and manage the development of software typically using less-abstract-models that target a particular software manifest. Modelling has a crucial role in Model Driven Engineering (MDE) methodology since MDE harnesses greatly the strength of modelling to improve engineering processes [47].

The process of modelling itself often starts from real world objects—a concrete experience—and then applies a process of abstraction [48, 4]. Typical modelling architectures have four layers. The first layer is real world object, the second layer is the model, the third one is the metamodel that defines the language of the model, and the fourth one is the metametamodel [47]. 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 regarding themselves, to remain within four levels of abstraction, rather than 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 of a flowchart language used to the waking-up process. The metamodel constitutes the definition of a language that defines the model since it has rules to describe the model represented by the language [47]. So, every model created should conform to its metamodel. The metamodel is more general than it is models since it could be

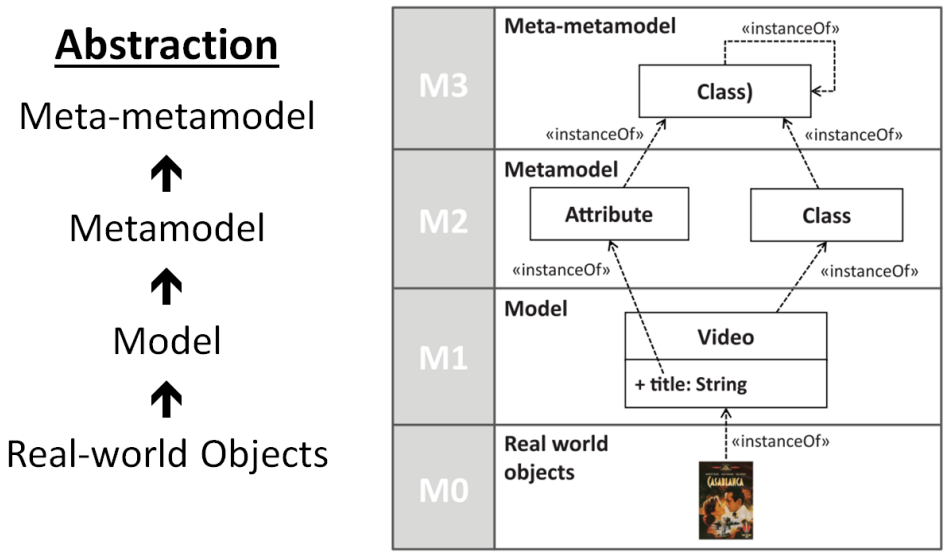


Figure 2.10: Abstraction in Model-driven Engineering [47].

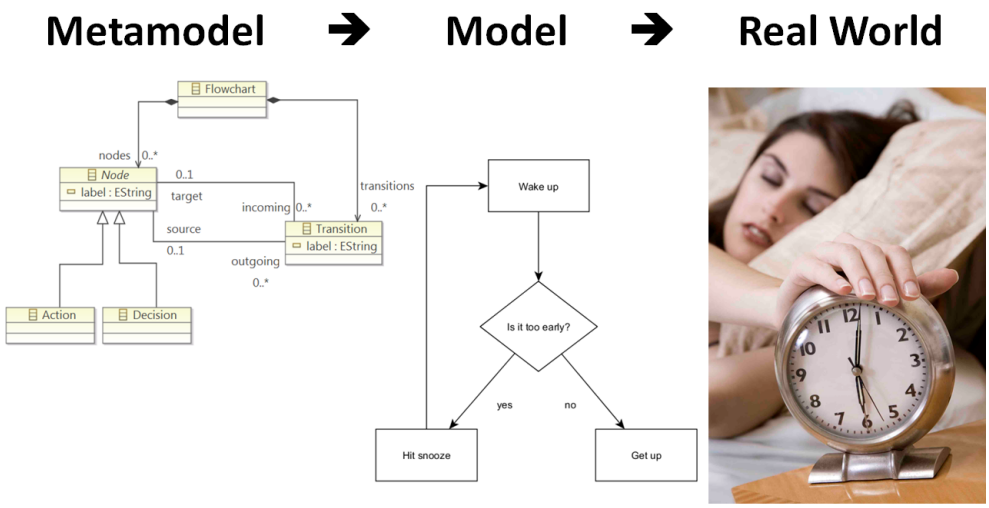


Figure 2.11: Instantiation in Model-driven Engineering.

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.

2.3.2 Software Modelling Tools

The use of tools for software modelling is encouraged since with them we can easily manipulate models, create models that at least conform to their metamodels, and perform model transformation [47]. 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 [49] to enterprise-scale modelling suites [3].

For the purpose of learning, Dranidis et al. [49] and Akayama et al. [51] suggest developing simplified software modelling learning tools, since, for beginners, modelling using professional tools can be overwhelming. 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 [50]. The model management itself consists of several processes. They are code generation, model-to-model transformation as well as model validation, comparison, 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 is dedicated to producing graphical modelling editors automatically[52].

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

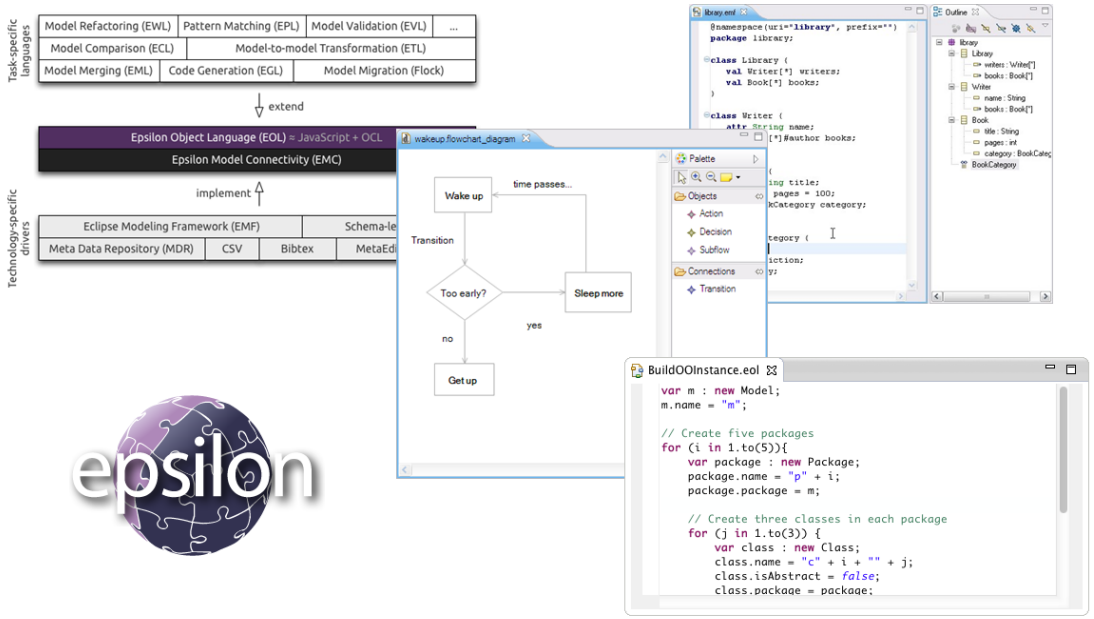


Figure 2.12: Epsilon is a family of languages and tools for model management [50].

2.3.3 Software Modelling Teaching

We have investigated related literature regarding the teaching and learning of software modelling. From the literature, we have identified findings and categorised them into three groups: contents, teaching or learning practices, and tool design. The findings are to be considered as requirements for the design of SMLG.

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 address software modelling definition at first [53] and then teach modelling foundation, such as semantics, syntaxes, and notation [53]. However, the teaching should focus on semantics, not only syntax [53], and improving the use of language, not just vocabulary [3].

The core, important concepts of Model Driven Engineering—modelling, metamodelling, and model transformation—should be taught. Teaching modelling should comprise the following topics: formal and informal models, partial and complete models, and the distinction between models and programs [3]. Nevertheless, metamodelling should be given a larger portion than modelling since a model might conform to more than one metamodel [54].

Software modelling is an engineering subject. Therefore, we should teach the engineering aspects of software modelling, such as understanding a domain, planning and resourcing, documentation, quality, formality, validity, and optimisation [55].

Another important topic in teaching software modelling is the practices or

applications of software modelling in various domains [53, 55], including their success stories, code generations, model discovery, and model-driven interoperability. It's best to make model executable, since the power of execution makes it much easier to understand the model [3].

Teaching and Learning Practices

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

Modelling is a process to think abstractly about systems. Therefore, modelling is thaught to make students understand the value of abstraction [3]. Additionally, successful application of Model-Driven Software Development depends on abstract modelling skills [2]. Another best practice suggested is that software modelling should be taught with prerequisites [55]. Therefore, the learners should have a good programming background [3] or know a little about OOP [51]. However, for an introduction to modelling in computer science/software engineering programmes, we can teach modelling alongside with programming[53, 3]. In this way, students can learn to model as early as possible [51, 53].

In teaching software modelling, we should encourage students to produce "good" models and measure their "quality", therefore they will be informed how good are their models. One way to measure the quality of models is using tools [51]. In teaching software modelling, problem-solving should be taught first, while modelling language specification and modelling tools can get in the way [55]. Regarding the problem-solving approach, educators should give solutions, not just direct answers [55]. Another best practice is that educators should teach modelling language broadly, not deeply [55], and throughout [53]. 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 [51]. Consequently, teaching should refer to other disciplines or other aspects related to software modelling as well [55].

Even though code generation is essential to understand modelling, since it shows the real-world application of Model-Driven Engineering, educators need to teach other applications (benefits) of software modelling at first [56]. The code generation could come later [55]. Also, the educators should be careful when using analogies and physical decomposition since they might not reflect the complexity of the system; one component might have a cross-cutting effect to other layers of the system [55]. Furthermore, it is good to teach modelling with a standard language, such as UML [3]. However, educators need to teach modelling and metamodelling using other modelling languages as well, not just UML. Software modelling is not a UML modelling course [55]. 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]. Choosing a playful domain or fun problems, not serious domain, is suggested to increase learners' engagement [55].

Tool Design

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

We can teach software modelling using papers, tools designed for pedagogy, or mainstream tools [51]. Each comes with its advantages and drawbacks. For an introduction, paper is a good choice. The use of modelling tools is not important in the beginning, but later when the modelling task becomes larger, more advanced tools could be considered [3]. However, learning modelling tools is not trivial [55]. Considerable efforts required to become fluent using a tool. A good experience with a tool has a positive influence on learners' view on modelling [56]. Likewise, support for tool usage also affects student satisfaction greatly. So, if industry modelling tools are used, it is critical to provide adequate support, such as providing access to a tool expert and carefully designing the instructions [56]. For example, Papyrus is suitable for a classroom environment if it is given with the right level of support [56].

Based on his experience developing an educational software modelling tool, Lethbridge [57] proposed the following best practices to produce a good educational modelling tool: (1) build knowledge and skills incrementally, (2) give positive reinforcement to learners, (3) the tools offer maximum opportunities for learning, (4) the tool convinces learners of the value of what they learn, (5) focus on high usability, and (6) the tool is inexpensive.

2.3.4 Abstraction in Software Modelling Learning

The review of software modelling and its teaching leads us to conclude that modelling and meta-modelling, and additionally model management, are potential concepts to be taught in the game. All three concepts require adequate abstraction skill, a skill that we have shown to have a fundamental role in computer science and software engineering [4, 5, 6]. There are some available strategies for teaching abstraction. One of them is the Familiarity, Similarity, Reification, and Application steps proposed by White and Mitchelmore [48] for mathematics. They argue that abstraction should be developed from empirical experience to abstraction. The abstraction then is reified to become an application. Similarly, based on their experience teaching UML, Engels et al. [4] approach abstraction through modelling following three 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] proposes three methods in teaching abstraction. First, illustrate: the lecturer uses abstraction-related words or statements in teaching. Second, reflect: for example, the 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 the use of abstraction.

There are two approaches to learning software modelling which learners may perform simultaneously: a rational approach and an intuitive approach [58]. Usually, at the beginning of the modelling process, learners use their existing knowledge and

logical reasoning to understand and develop software models, sometimes preceded by empirical activities. However, there are situations in software modelling where existing knowledge and methodology-based, logical reasoning are not enough to allow learners to understand or to develop a software model. In this situations, the learning might expect the learners to rely on their intuition, might present a combination of both rational and intuitive approaches. Moreover, in building a model at an appropriate level of abstraction, one needs a level of intuition and skill which cannot be gained through lectures alone, but has to be experienced through exercises [4]. This approach is also encouraged by experiential learning [21].

2.3.5 Gamified Software Modelling

Most of the gamification studies available related to software engineering in a wider 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 Stikkolorum et al. [59], Ionita et al. [60], Groenewegen et al. [61], and Richardsen [62]. findings from these related studies are presented in section 2.4.

Puzzle Game to Teach Software Design Principles

Stikkolorum et al. [59] develop a game that is intended to teach software design principles, such as cohesion, coupling, information hiding, and modularity in object-oriented software design. They look for a solution that provides a balance between coupling and cohesion by using a toolbox to draw classes, methods, attributes, and relationships. They apply game elements like puzzles, game levels, visual and audio feedback, a progress indicator, level unlocking, choice of paths, multiple solutions, and scoring. For the pedagogical aspect, they refer Bloom's taxonomy in their work. However, they did not explain how they integrated the taxonomy into their work. They validate their design by conducting user test and utilised the 'think aloud' method, asking users to tell their thoughts while using the game. After using their game, Stikkolorum et al. [59] observe that users start to refer to concepts such as classes, methods, and associations instead of boxes, blocks, and lines, indicating a progress of learning. The main challenge of their work is related to scoring since there are many valid solutions for a problem. To determine coupling, they use the Coupling Between Object Classes (CBO) metric [59]. Cohesion was measured by comparing all items—attributes, methods, class name—that have similar keywords under the same class. Information hiding and modularity is evaluated using general design patterns.

Explorable Board-game to Understand and Validate Enterprise Architecture

Groenewegen et al. [61] apply gamification to improve stakeholders' understanding of their enterprise architecture models as well as to validate them. They employ

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. They claim that users can understand the model better than by merely looking at it, so they can argue whether the model is valid or not based on their existing knowledge. However, they do not give any pedagogical reason of the claim. The challenge that they experience 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 makes a model less understandable, and users cannot validate the model. For validation, they test 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.[60] develop a socio-technical modelling language (TREsPASS) that maps information on 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. The tangible representations are claimed to increase the familiarity and understandability of models, which could 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 an experimental group performed better than a 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 [62] develops a game, whose behaviours are controlled through an UML activity diagram. Throughout his research, he identifies a challenge in controlling the game from the activity diagram in the Reactive Block Environment[4], an eclipse-based visual development environment for Java application that is difficult since Eclipse[5] is difficult for a first-time user. For validation, he conducts user testing with three users. The 'think-aloud' method is used for observation. After that, questionnaires are given, and an interview is conducted. He concludes that there is no significant difference between the traditional

[4] http://reference.bitreactive.com/
[5] http://www.eclipse.org/home/index.php

interactive tutorial and the game-like tutorial on their performances. However, the game-like tutorial is more engaging. For the validation of his work, the author does not mention any explicit pedagogical aspects.

2.4 Synthesis

So far, we have presented studies categorised in pedagogy, ludology, and software modelling fields. In this subsection, we summarise the related studies and present findings and the plan briefly for how we are going to integrate the findings into our research.

Pedagogy. In the pedagogy domain, we have presented learning models that we plan to apply in our design, including the theory of Flow, Scaffolding, Bloom's taxonomy, experiential learning, and motivational learning. Each model has its contributions to the design of our game. Theory of Flow tells us that in order to maintain learners' motivation, their competence and given challenges have to be kept balanced [15]. We plan to use the theory to guide the design of levels, balancing challenges and learners' competencies. Scaffolding suggests us to give reinforcement to learners to help them grasp difficult but important concepts [19, 20]. We plan that our SMLG should also provide scaffolding to learners. It should present helpers and cues to learners to help them solve problems and grasp key concepts. The scaffolding will be removed gradually as the competence of the learners grows.

Moreover, Bloom's taxonomy provides us six activities with different cognitive loads [16]. We see the potential of the six activities of Bloom's taxonomy: the activities that the learners will perform when playing the SMLG and the nature of cognitive load of each activity could be applied to the challenges of each level. Thus, Bloom's Taxonomy will give us a variety of options—activities and challenges—in designing the levels of our game. Kolb's model gives us a framework to design a learning activity cycle for experiential learning [21]. We plan to apply the model to the design of our SMLG so it will allow learners to perform active experimentation and develop their understanding of software modelling and make connection between their knowledge and real experiences. The motivational model provides us four components that we should consider to maintain learners' motivation [23]. The model to be applied to the design of flow of a level of a SMLG to maintain learners' motivation. We will elaborate and discuss these learning models and our design in more detail in section 4.3.

Ludology. In this report, we will use the term software modelling learning gamification (SMLG) to represent the products that are the target results of our gamification processes. To guide our SMLG design, we choose Gameful Design framework [35], since we judge the framework stimulates innovative gamification ideas arose from stakeholders' needs and contexts through a deep thinking about the interconnection between game elements and core activities that are going to be gamified. Up to now, we have followed the prescriptive guidelines of the Gameful Design framework to generate the preliminary design of our game and developed a storyboard to create a visual representation of our design (Appendix E). We also

have performed our preliminary survey to identify learners' needs, motivations, and challenges (section 4.1). The result of our initial design and its early prototype can be found in section 4.4.

Software Modelling. In sections 2.3.4 and 2.3.3, we have investigated studies related to abstraction in learning, specifically in software modelling, and best practices in teaching software modelling. From these related studies, we derive several design requirements, which can be found in section 4.2. From gamified software modelling related studies in section 2.3.5, each study addresses different topics in software modelling. However, none of them addresses the core topics of Model-Driven Engineering—modelling, meta-modelling, and model transformation, which means that there is an opportunity for novel research in the area. We also found that each study addressed its topic with different approaches and game elements, which also challenges us to develop a more generic design in addressing software modelling learning problems. Moreover, the common drawbacks of the studies are that most of them did not consider the pedagogical aspect of their solution and their validation was weak in sample size as well as the lack of discussion of internal validity. Nevertheless, all of the studies reported that their gamified approaches have a positive effect—it is motivating and engaging users in varying degrees, which confirms that gamification has a positive impact on motivation.

Still from gamified software modelling related studies in section 2.3.5, we also identified some constructive findings to improve the quality of our research and SMLG design. First, the quality of models created by learners has to be measured to give them feedback how good the models are. Software metrics could be applied to measure the quality. Second, evaluation should conform to the standard criteria of good research practices in terms of sample size and internal validity. Third, pedagogical aspect should be integrated into the SMLG design.

Output Tool and Framework. Simplified software modelling learning tools for learning purpose are appropriate for beginners since modelling using professional tools can be overwhelming [49, 51]. Thus, we plan to build our SMLG as a tool that is simple, familiar, and accessible. Simple means the SMLG focuses on core activities so learners can easily learn software modelling. Focusing on core activities also satisfies the requirement of gameful design [35]. Familiar means the SMLG mimics common visual modelling environments that usually take in the form of a window that has a palette and a drawing area. Accessible means the SMLG is a web-based application so learners can access the SMLG as long as they have an internet connection.

We also plan to build a framework for SMLG to ease the design and automate the generation of the SMLG. The design framework will be built upon the Epsilon family, including the utilisation of other technologies such as Javascript, HTML, and J2EE to create web-based SMLG. Epsilon is a family of languages and tools for model management [50]. The model management itself consists of several processes. They are code generation, model-to-model transformation as well as model validation, comparison, migration and refactoring. Moreover, Epsilon also supports Eclipse Modeling Framework (EMF) and other model types, such as Meta Data Repository (MDR), CSV, Bibtex, etc. (Figure 2.12). A visual modelling environ-

ment for the design framework will be generated using EuGENia [52], a subset of Epsilon family intended to produce visual modelling editors automatically.

Chapter 3

Proposal

Grounded on the literature review in Chapter 2, in 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 by our research and the research methodology that we are going to apply to address the research questions.

3.1 Research Questions

The main research question proposed by this research is "Can gamification improve software modelling learning?". The word 'improve' implies that learning with gamification enhances learners' engagement and learning performance. Learners with the support of gamification engage more durable, frequent, and active compared to learners that only use didactic approach. Also, the former ones perform better in knowledge and skill acquisition and application compared to the latter ones. To answer the main research question, following sub research questions need to be investigated:

1. Which parts of software modelling teaching and learning could benefit from gamification?
2. What teaching and learning best practices of software modelling that are significant to be accommodated in SMLG?
3. What pedagogical learning models and in what roles that can improve the engagement and effectiveness of SMLG?
4. What kind of gamification design that can support software modelling learning best?
5. What kind of orchestrating framework is needed to design the interaction between software modelling and game elements to achieve effective SMLG?
6. To what extent does SMLG improve learners' motivation, engagement, and performance?

7. To what extent do software modelling tutors benefit from a SMLG 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 an SMLG framework that systematically and semi-automatically drives gamification design and generation to produce SMLG. 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 framework that is intended to design and generate SMLG based on the literature review and survey. The framework will be iteratively updated according to the results obtained from experiments.
3. Design and generate instances of SMLG. The instances will be tested to respondents for evaluation and to obtain feedback for iterative improvement.
4. Perform controlled experiments to measure the significance of the SMLG in improving learning performance compared to traditional method, didactic learning without the support of the gamification.
5. Perform controlled experiments to measure the productivity and maintainability of a software modelling learning design framework in supporting tutors design and develop SMLG.

3.3 Research Outputs

The potential research outputs of this research are:

1. Software/applications of SMLG. The applications/instances of SMLG that are designed and generated using a SMLG framework.
2. A framework for designing and generating applications of SMLG.
3. Controlled experiments: a learning outcome comparison between gamified version and the traditional one of software modelling learning.
4. A model that explains how SMLG works. The model can be achieved through Learning and Game Analytics and Structural Equation Modelling studies.
5. Case studies: reports of applications of theories, models, and methods used in this research.

3.4 Research Methodology

In this section, we discuss briefly the research methodology that we apply to address research questions presented previously. Since the main outputs are design artefacts—the applications of SMLG and the SMLG framework to generate them, Design Science Research Methodology [63] is selected as the research method as it provides a comprehensive conceptual framework and activity guidelines for understanding, developing, executing, and evaluating the design artefact. We also discuss briefly our evaluation plan to evaluate our research findings. In the end of this section, we present our Research Data Management approach as part of good research practices.

3.4.1 Design Science Research Methodology

We will employ Design Science Research Methodology [63] 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 3.1). Another reason is that it positions itself at the top level of abstraction without going into much detail of how to perform each activity, we can freely choose other more concrete research methods to carry out the activities. For examples, we can conduct literature reviews, surveys, or expert interviews to determine research problems, motivations, solutions, and objectives as well as controlled experiments to measure and evaluate the effectiveness of the artefacts.

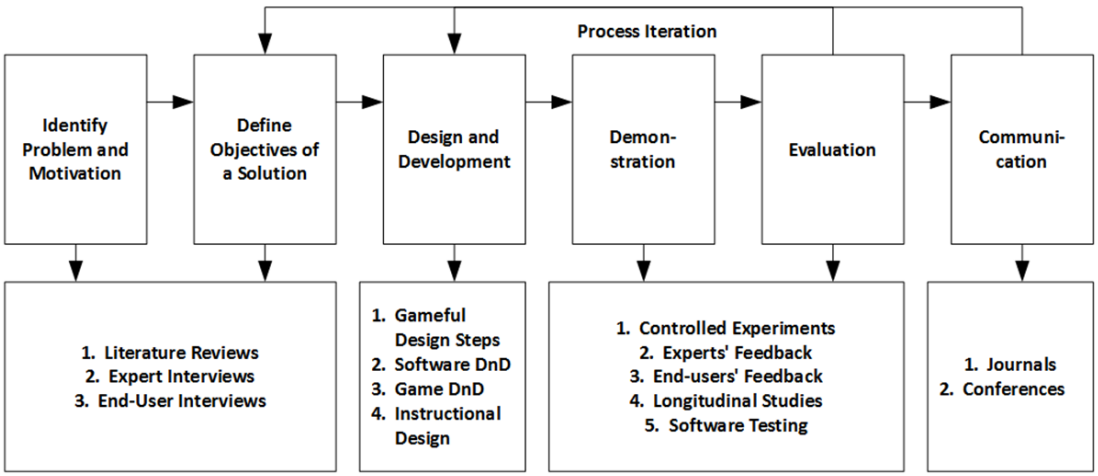


Figure 3.1: Design Science Research Methodology. Adapted from Peffer et al. [63].

Identify Problem and Motivation. This research will use literature review, suggestions from experts, and surveys to identify research problems and motivations as the background to determine the solution and its objectives. The experts are academicians or practitioners that have significant experience in Model-Driven Engineering and respondents are the students of Model-Driven Engineering-like modules (Software Analysis and Design, Model-Driven Software Development, etc.). In this

way, (1) we can identify parts of software modelling teaching and learning that could benefit from gamification, (2) we can also determine software modelling teaching and learning best practices that are significant to be accommodated in SMLG, (3) we are facilitated to decide which pedagogical learning models that can improve the engagement and effectiveness of SMLG, as well as (4) to determine what kind of gamification design that can support software modelling learning best. Performing problem and motivation identification addresses research questions one to four.

Define Objectives of a Solution. Based on the identified problems and motivations, we define our solution that is embedding gamification in the process of software modelling learning, which is expressed by the use of gamification applications specifically designed for software modelling learning. Research objectives have been defined in the previous section (section 3.2).

Design and Development. This research will employ Gameful Design Framework [35] to design and develop the SMLG and agile software development [64] to design and develop the software/applications. 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. For the framework that generates the SMLG, we build one application of SMLG that addresses one graphical modelling language. From there, we make an abstraction of reusable components of the application and implement Model-Driven Engineering approaches, such as modelling, metamodelling, and code generation, to generate different applications specific for other graphical modelling languages.

Demonstration and Evaluation. The resulting applications and the framework to generate them will be demonstrated and evaluated by applying it to several courses of software modelling. Moreover, the evaluation results will be used as feedback to improve the quality of the applications and the framework 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.

3.4.2 Evaluation

We plan to evaluate (1) the effectiveness of the SMLG discussed above and (2) the productivity and maintainability benefits of the SMLG 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 SMLG. 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 SMLG framework, the participants will be software modelling tutors; they will be divided into two groups, one that will develop SMLG *with* the SMLG framework and one *without* the SMLG framework (i.e. using ex-

isting web technologies). They will be asked to elaborate their SMLG into their teaching instructions and use them in their teaching. The comparison will be on their productivity and the maintainability of their SMLG. 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 [65] 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 SMLG's event logs using data mining or machine learning techniques.

3.4.3 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[1], 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 [66]. To support our work, we will use the DMP template[1] provided by the University of York for postgraduate research projects.

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

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 SMLG. In the end, we present the preliminary design of the SMLG as well as the SMLG framework.

4.1 Preliminary Survey

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

We have distributed online questionnaires to students of Model Driven Engineering (MODE) 2015/2016 module. The students were in their Software Engineering master programme at the University of York. From 21 students, only 4 completed the questionnaires. Their responses can be found in Appendix C. Since the number of respondents are small for generalisation, we plan to apply the same survey to next term MODE students.

Results. To identify the learners' needs, we asked our respondents two questions. Question 1 aims to identify students' expectations before starting the module, while question 2 aimed at identifying what the students found important after taking the module. Based on the responses, the reasons why the students took MODE module because they want to increase their knowledge on MDE, possess new advanced skills or abilities and improve their literation of MDE tools. After completing the module, the students valued that the most important lessons were

getting new knowledge—domain modelling, metamodel, abstract syntax, abstract thinking, model validation, and the application of models—and skills—generating code, creating DSL, and improving their tool skills.

We also asked the students three questions (Q3-Q5) to identify their motivation in taking MODE module and Learning Model Driven Engineering. Question 3 asked about the reasons behind their decision taking MODE module. Two students stated that they took the module because it is compulsory, but the rest of the students said that they took the module because MDE is an advanced topic and they wanted to see its applicability in the industry and whether it will improve their ability—knowledge and skills.

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

We then asked question 5 which asked the students the most basic, underlying motives that make them commit to learning MDE. Substantially, they answered that their main motivation is to gain new ability—knowledge and skills, such as the ability to make an abstraction, advanced skills that are applicable in industry, and knowledge of real-life examples and applications of different models taught in MDE. Nevertheless, passing MODE module, a pragmatical motive, still part of the whole motivation.

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

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

Discussion. There are few findings from the preliminary interview regarding students' needs, motivations, and challenges, that can be implemented into the design of a SMLG. *First*, the need of the students to learn MDE is to gain new advanced abilities—knowledge and skills—that are applicable in industry, which, if

broken drown, they comprise of model management activities (abstract thinking, modelling, metamodelling, model transformation, validation, and application) and tool literacy.

Second, the motivations of the students to learn MDE are, regardless their view on MODE module as a compulsory module in their programme, they were aware that their motivation should be on satisfying their need in gaining advanced knowledge and skills in MDE as mentioned before, and they would be more motivated if they could perceive the advantages and applicability of MDE, thus showing students the benefits and applications of MDE are crucial in increasing their motivation.

Third, they mentioned model management activities (abstract thinking, model validation, metamodelling, etc.) as the interesting challenges. To overcome the challenges, they did trial-and-error method to gain more experience in overcoming the challenges. These findings are in line with experiential learning which states learning is best achieved through experiencing [21]. 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 [35].

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 [56].

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 2.3.3. These requirements have two roles. First, they provide guidance to our design process and, second, they will also act as units of evaluation to confirm the SMLG meets the current best teaching practices.

We also derive other requirements from our preliminary survey in section 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 SMLG. 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 4.1: 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).
Uninteresting Challenges	RS06	Increase the usability of the tool being used.
	RS07	No need to learn the detailed technical terms specific to certain products.
	RS08	Provide documentation and support for the tools being used.

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 SMLG. In this section, we will explain the relationships between the learning models, their contributions, and how they will be applied to the design of the SMLG are depicted in in Figure 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 SMLG. We translate challenge into series of levels in our design and of course higher levels come with higher difficulty. To realise this, we borrow the learning activities—remember, understand, apply, analyse, evaluate, create—from Bloom's taxonomy, since every activity has different cognitive loads according to their order with 'create' has the highest cognitive load. We assume that activity with higher cognitive load is also harder to complete. Therefore, we can make different combinations between the activities that will gradually increase in difficulty (cognitive load) along the increase of the levels (Figure 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.

While learners are progressing in the SMLG, they are developing their competence. Thus, difficulty has to be kept balanced with their competence, otherwise they will get bored. It is the situation where the theory of Flow can be applied (Figure 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 in-

Table 4.2: Requirements derived from the literature review (section 2.3.3).

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

troduction 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 [23]. The model provides us in each its components—attention, relevance, confidence, satisfaction—a set of predefined techniques to maintain learners' motivation. In a course of a level of SMLG, there are a start, an end, and learning activities in between (Figure 4.1). We could apply the ARCS' techniques to maintain learners' motivation along the course of completing a level. As an example, we could use animation to gain learners' attention, explaining the application of the concept being taught in the currently playing level to give relevance, showing their progress in completing a level to maintain their confidence, and giving them a reward after finishing a level for reward.

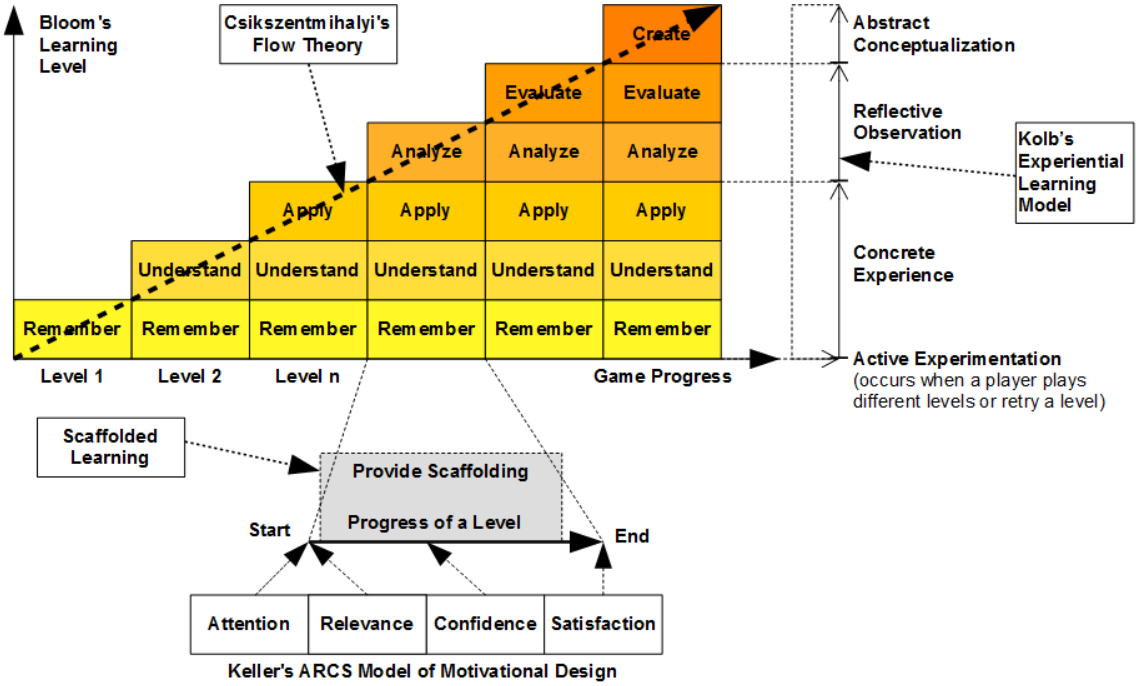


Figure 4.1: Elaborating learning models' contribution to the design of the gamified modeling learning

.

We apply scaffolding [20, 19] 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 applying Kolb's experiential learning model, which is a model that agrees knowledge is constructed through experience and based its model on constructivism [21]. We select Kolb's model since we perceive that playing a level in SMLG is similar to the learning cycle Kolb proposed; a cycle consists of 4 steps: concrete experience (CE), reflective observation (RO), abstract conceptualisation (AC), and active experimentation (AE).

We could apply this cycle to frame learners' activities in gaining new knowledge through solving a problem given in a level. For example, the first time players play a new level, at that moment they encounters a concrete experience (CE). Immediately, they attempt to identify and characterise the problem given in that level and recall any knowledge that is relevant to solve the problem (RO). Next, they construct a solution for the problem that they face (AC). After constructing the solution, they apply the solution to the problem (AE), experience the result (CE), and then evaluate whether the solution solves the problem of the level or not (RO). Any gap that appears will update their knowledge. They use their newly updated knowledge

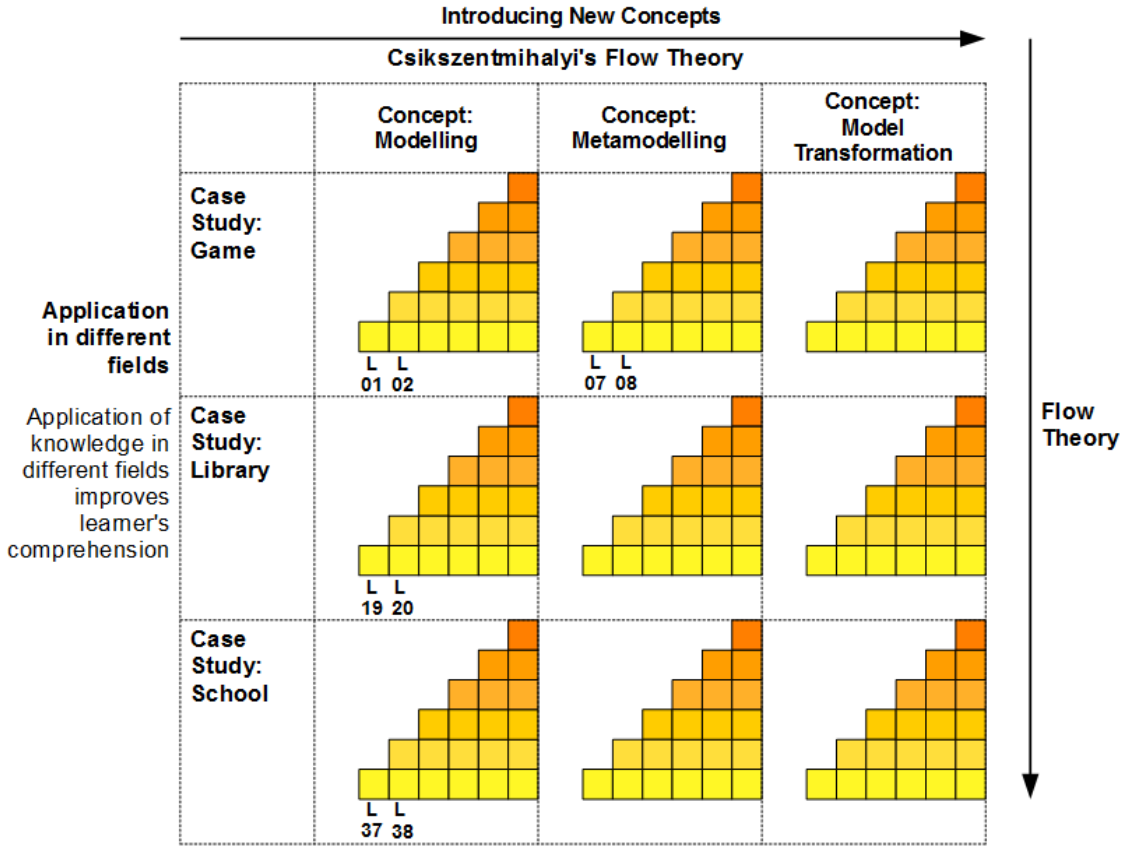


Figure 4.2: Elaborating learning models' contribution to the design of the gamified modeling learning

.

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.

Learning activities in Bloom's taxonomy also correspond to the steps in Kolb's learning cycle [67] and both have been applied together to design instructions in different fields [68, 69, 70]. 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 Gamification Design

In this research, we plan to assess whether gamification is beneficial for learners of graphical software modelling languages. We choose graphical modelling languages since they are the common languages used in modelling, whether in academia or industry, and extensively used in Model-Driven Engineering. Standard graphical

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.

modelling languages like UML[1] and BPMN[2], often used in Model-Driven Engineering, are some of the use-cases.

Modelling can be expressed in different modelling languages. To minimise bias and ensure the generality of our SMLG, we plan to experiment and support several graphical modelling languages (e.g. UML, BPMN, state-charts). For each modelling language, we envision the development of dedicated SMLG that will be derived from the Gameful Design Framework [35]. The SMLG will mimic a graphical modelling tool, and at each level, it will require the learner to graphically construct or adapt a model to meet a set of constraints and requirements.

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

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

A real-world problem can be time-consuming and very complex to model. Thus, the inessential activities that are not significant to the core concepts that are being taught should be excluded. As a result, learners will be more focused on the main concepts. Thus, game elements like limited choices (i.e. only limited items can be dragged), microflows (i.e. put the right element to its right place), and bite-sized actions (e.g. drag and drop) will be implemented to facilitate learners in performing the core activities. Likewise, fuzziness will also be used to stimulate learners' creativity since most of the time there is no single correct model for the problem at hand. Attractive design will also be significant to motivate learners to interact with the SMLG. The SMLG should be able to give instant, noticeable, and

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

actionable feedback to maintain learners' engagement and monitor their progress. Interesting and varied feedback should be designed to appeal to the learners' motives. We also plan to implement the SMLG using web technologies so that they are accessible to a wide audience.

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

4.5 Gamification Design and Generation Framework

We plan to build a framework that will facilitate the design and generation of SMLG. Rather than developing SMLG for each graphical modelling language manually, we will follow a model-based approach. In the spirit of Eugenia [52], we will use metamodel annotations to define the graphical syntaxes of modelling languages and separate models to specify the game elements (constraints, objectives, levels, etc.) of the SMLG. These models will be then consumed by a model-to-text transformation to produce fully-functional language-specific SMLG. Thus, the framework supports software modelling tutors in the design and customisation of the SMLG at the high level of abstraction as well as to automatically build the SMLG. Up to now we have implemented a metamodel to specify game elements (flows, levels, challenges, and objectives) and a supporting Eclipse-based graphical editor (Fig. 4.3), and a prototype SMLG (Fig. 4.4) for object diagrams.

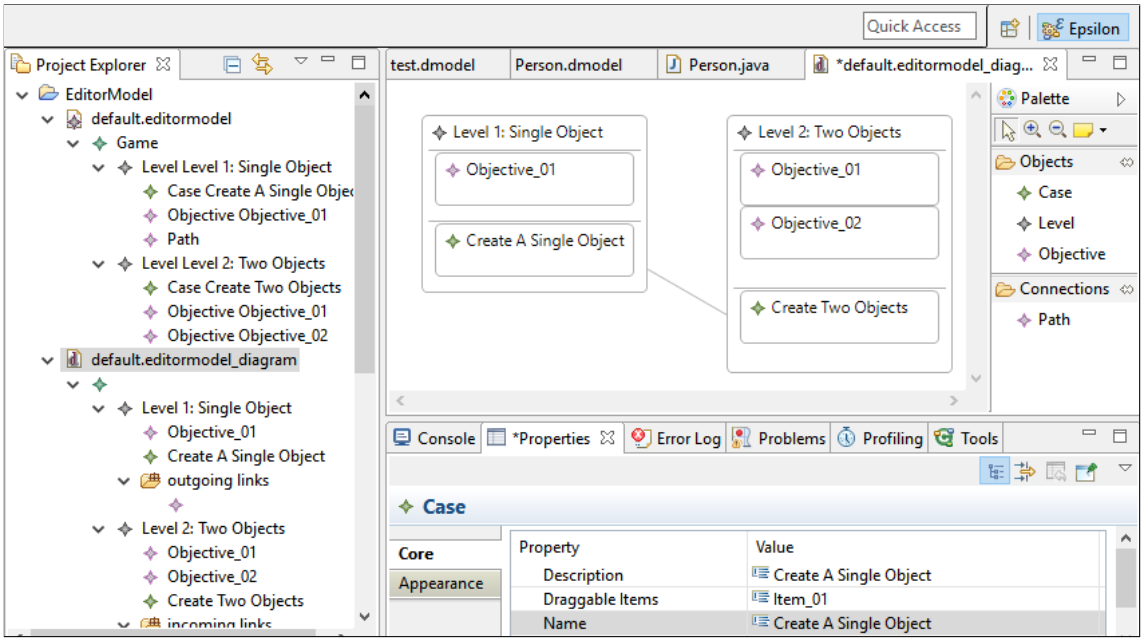


Figure 4.3: Graphical editor for the gamification specification DSL [71].

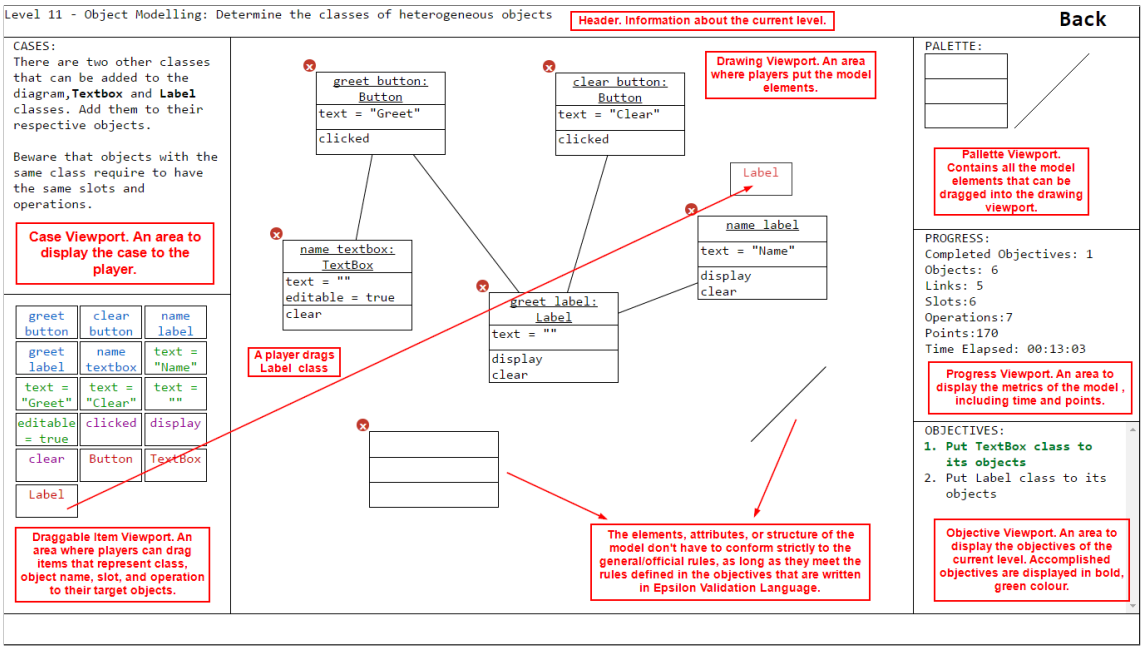


Figure 4.4: The display of the generated game [71].

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] N. Fleming and D. Baume, "Learning styles again: Varking up the right tree!" *Educational Developments*, vol. 7, no. 4, p. 4, 2006.

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

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

[17] H. Gardner, *Frames of mind: The theory of multiple intelligences*. Basic books, 2011.

[18] R. M. Felder and L. K. Silverman, "Learning and teaching styles in engineering education," *Engineering education*, vol. 78, no. 7, pp. 674–681, 1988.

[19] 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.

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

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

[22] R. M. Ryan and E. L. Deci, "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being." *American psychologist*, vol. 55, no. 1, p. 68, 2000.

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

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

[25] 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.

[26] 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.

[27] 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.

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

[29] S. Arnab, T. Lim, M. B. Carvalho, F. Bellotti, S. Freitas, S. Louchart, N. Suttie, R. Berta, and A. De Gloria, "Mapping learning and game mechanics for serious games analysis," *British Journal of Educational Technology*, vol. 46, no. 2, pp. 391–411, 2015.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

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

[38] 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.

[39] 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.

[40] 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.

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

[42] Y.-K. Chou, "Octalysis: Complete gamification framework," *Yu-Kai Chou & Gamification*, 2013.

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

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

[45] W. H.-Y. Huang and D. Soman, "Gamification of education," Research Report Series: Behavioural Economics in Action, Tech. Rep., 2013.

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

[47] 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.

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

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

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

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

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

[62] O. Richardsen, "Learning modeling languages using strategies from gaming," Master's thesis, Norwegian University of Science and Technology, Norway, 2014.

[63] 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.

[64] T. Stober and U. Hansmann, "Overview of agile software development," in *Agile Software Development*. Springer, 2010, pp. 35–59.

[65] 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.

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

[67] 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.

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

[69] 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.

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

[71] 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

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

Table A.1: Research Timetable

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

Appendix B

Publications

We have published papers in the following conferences or journals:

1. A. Yohannis, "Gamication of software modelling, in the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016) Doctoral Symposium. CEUR, 2016 [71].

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 challenges—the noninteresting challenges—that hindered or demotivated you in learning ModelDriven Engineering? Why?**

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

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

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

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

Appendix 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, challenges

User needs

- Master the software modelling.

Motivations

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

challenges

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

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.

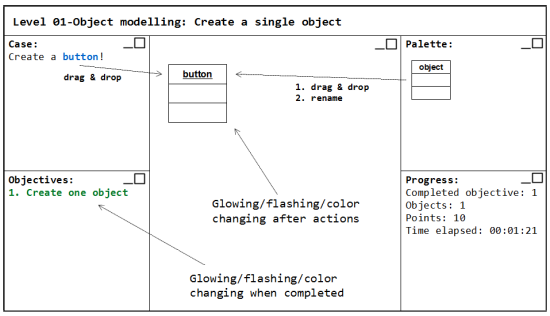
D.5 Iterative Prototyping

Prototyping is still on going work.

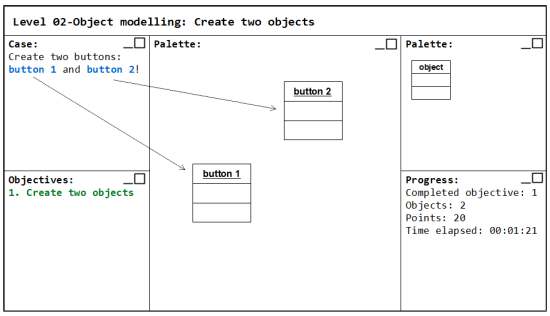
Appendix E

Storyboard: Object Diagram

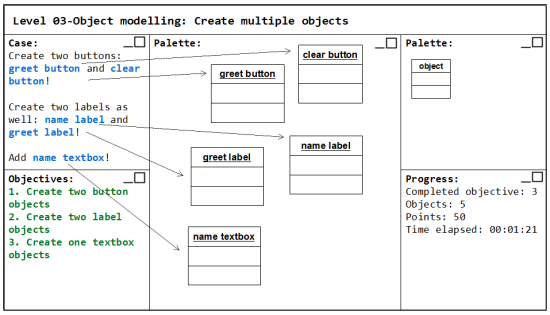
- Objects
  - Create single object (Level 01)
  - Create two objects (Level 02)
  - Create multiple objects (Level 03)
- Links (Relationships)
  - Create single link (Level 04)
  - Create multiple links (Level 05)
- Slots (Attributes)
  - Determine a slot and its value (Level 06)
  - Determine slots 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)



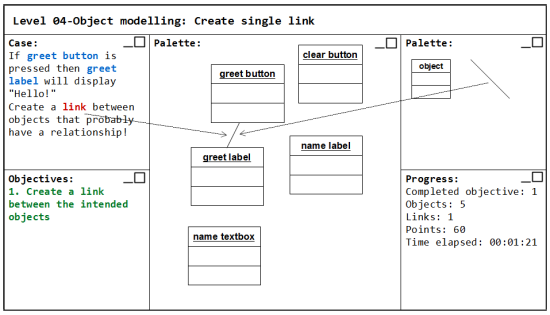
(a) Level 01



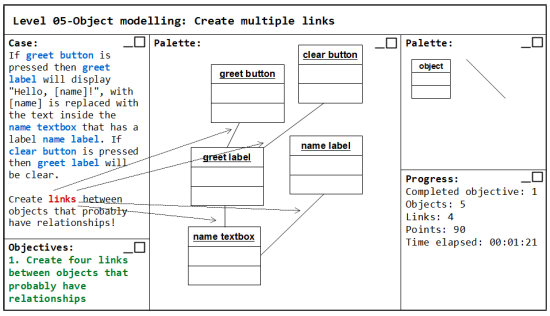
(b) Level 02



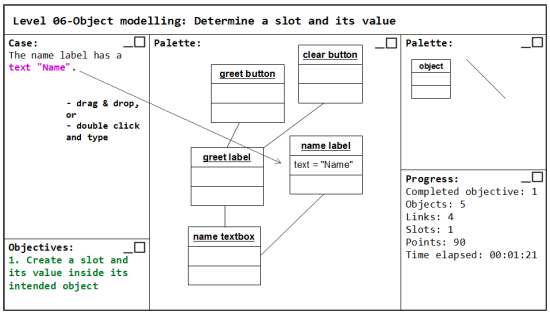
(c) Level 03



(d) Level 04

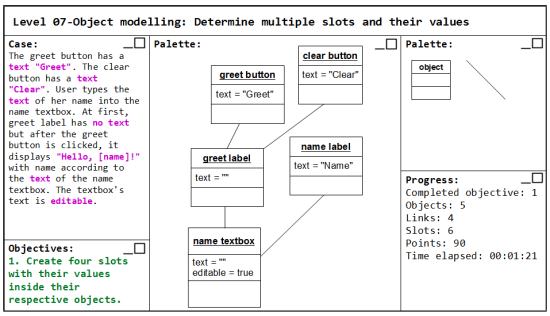


(e) Level 05

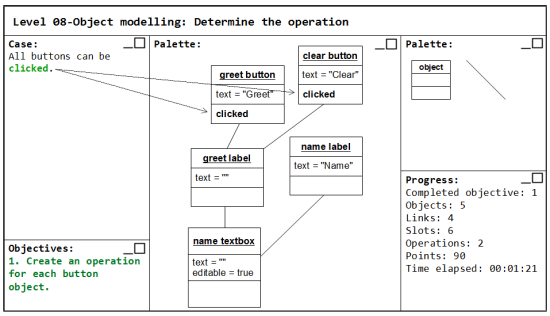


(f) Level 06

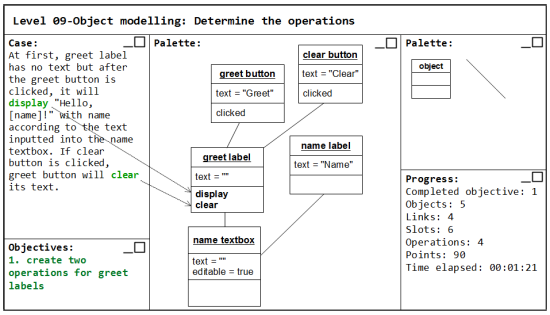
Figure E.1: Storyboard of Gamification of Object Diagram (part 1).



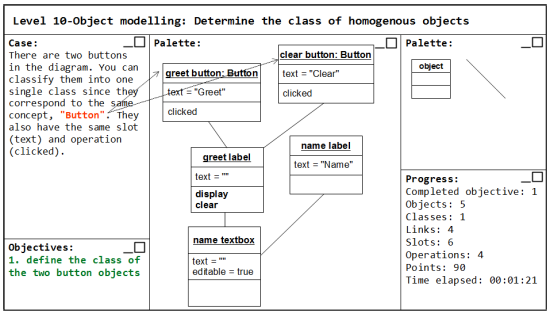
(a) Level 07



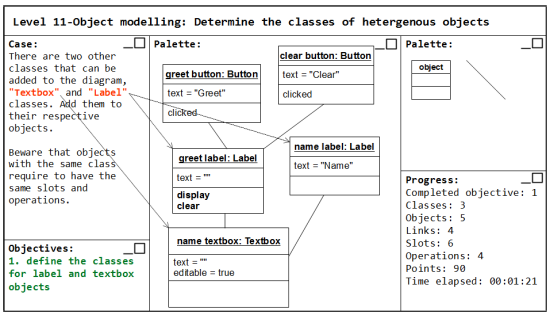
(b) Level 08



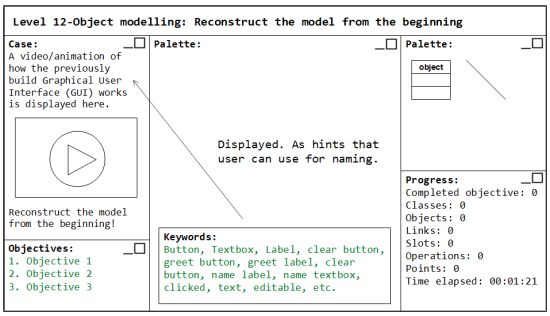
(c) Level 09



(d) Level 10



(e) Level 11



(f) Level 12

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