A Card Game for Learning Software-Refactoring Principles

Thorsten Haendler
University of Applied Sciences BFI Vienna
Institute for Information Systems and New Media, WU Vienna
thorsten.haendler@fh-vie.ac.at

ABSTRACT

While software refactoring is considered important to keep a software system maintainable and extensible, it is often neglected in practice due to several reasons. Besides the associated costs, software developers often perceive refactoring as a difficult and risky activity. However, apart from textbooks that document rules and best practices for identifying bad smells and applying appropriate refactoring techniques, learning and teaching refactoring poses multiple challenges. The application of these rules and techniques to code examples requires (advanced) skills in programming as well as an adequate handling of the programming environment. These circumstances can distract the focus and set barriers to the introduction of refactoring. In this paper, we present REFACTORY, a non-digital multi-player card game for learning principles of software refactoring without the development-related complexities. In our game, the players simulate a software-development team confronted with bad code smells. The players learn to combine refactoring techniques to remove the smells and to balance costs and value of refactoring. We specify the game design and illustrate the game workflow. In addition, experiences and lessons learned from a first game-play study are presented, e.g. regarding game fun and mediation of competences. Finally, we discuss limitations and further potential for improving the game.

Author Keywords

Card game, software refactoring, serious game, game-based learning

INTRODUCTION

Issues in software artifacts such as code, design or architecture can cause technical debt (TD) which can negatively impact a software system's maintainability and evolvability [24]. A popular technique for removing these issues (such as bad smells) is software refactoring, which aims at improving the internal quality while preserving the observable system behavior [30, 9]. But software refactoring is a complex activity involving several challenges and which is often perceived as difficult and risky by software developers [38, 12]. In recent years, only a few approaches in software-engineering education addressed the need for software developers competent in software refactoring (see e.g. [35, 17]). Most have in common that they require a high level of entry skills, e.g. regarding coding, software-design principles or handling of the programming environment (e.g. the testing framework).

For refactoring a certain kind of code smell, often different options and paths are available [9, 37]. Being aware of multiple (or all) options available (in best practices), gives the software developer advantages regarding flexibility and efficiency in problem solving since she then can choose the best (e.g. the easiest or most elegant) out of multiple options. Moreover, it is essential that software developers are aware of the value of software refactoring and are able to apply refactoring techniques in a meaningful and efficient way by weighing costs and benefits for the software project.

Educational games aim at fostering practical competences and motivation by providing a playful and interactive environment [26, 18]. As found out by [21], around 90% of games for software-engineering education are digital games, which means that they mostly provide a simulation of development activities and are played against the computer (for examples, see Section 2). In general, non-digital card and board games allow for focusing on selected conceptual aspects, have lower entry barriers and provide the benefits of face-to-face and informal interactions between players [21].

In this paper, we present a non-digital card game called REFACTORY for learning principles of software refactoring. In particular, the objective of the game is (1) to motivate the players for learning more about software refactoring and (2) to mediate basic refactoring-related competences in terms of being able to conceptually combine refactoring techniques to remove bad code smells as well as reflecting the balancing of costs and benefits of applying software refactoring. The proposed learning environment supports active and game-based learning, especially suitable for novices that have little experience in software development. By focusing on the conceptual level and certain aspects of refactoring, development-related complexities are avoided and the entry barriers are low. The basic idea is to confront players in the role of software developers with a software system that is affected by several bad code smells (technical debt). In multiple sprints, the players then aim at increasing the system's value by realizing functionalities. The players then have to decide whether to realize a functionality, which is more time expensive, when the corresponding component is affected by smells, or to remove

In: J. Arnedo-Moreno, C.S. González, A. Mora (eds.): Proceedings of the 3rd International Symposium on Gamification and Games for Learning (GamiLearn'19), Barcelona, Spain, 22-10-2019, published at http://ceur-ws.org

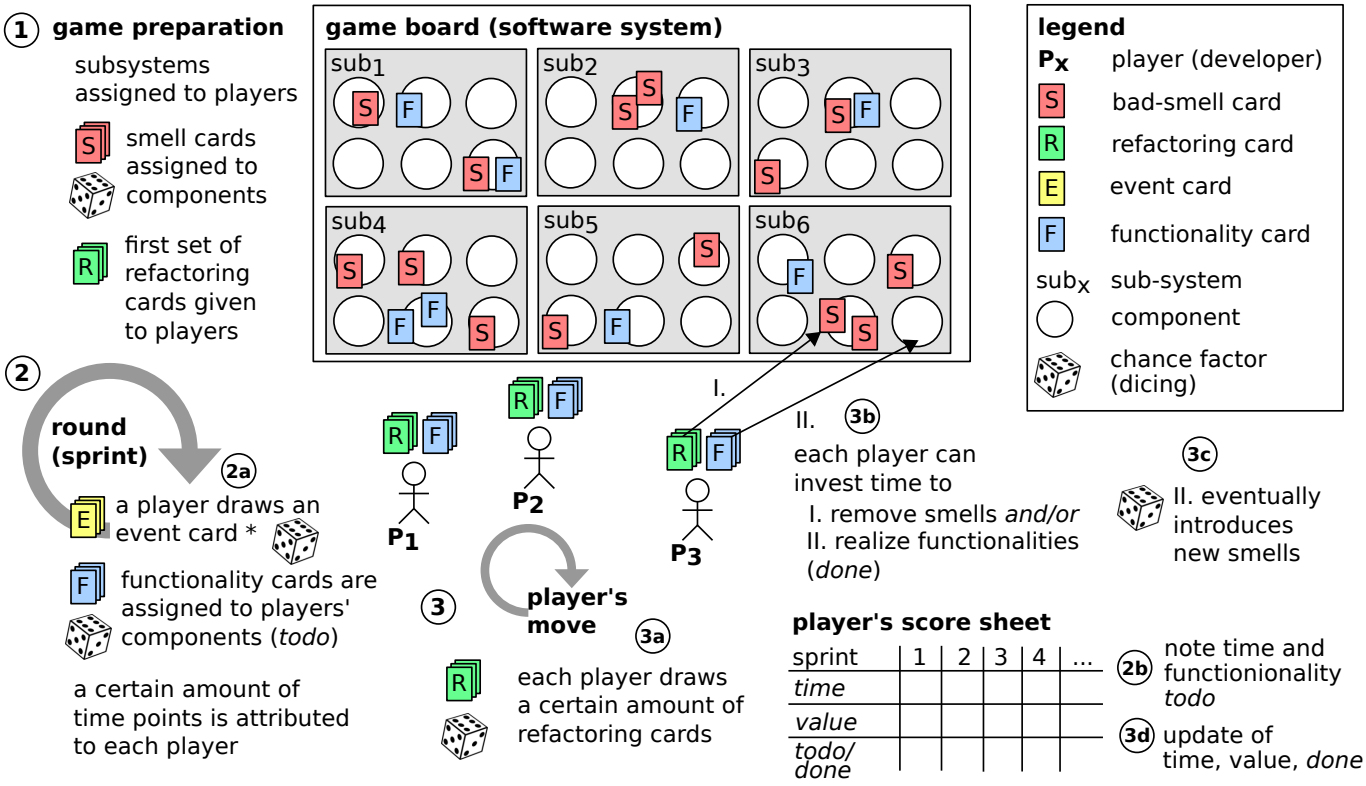

Figure 1. Game constellation of REFACTORY.

the smells beforehand by applying refactoring techniques. In particular, this paper provides the following contributions.

- We present design and workflow of an educational card game addressing basic competences in software refactoring, such as combining refactoring techniques to remove bad code smells and strategic aspects like balancing costs and benefits of performing refactoring activities.
- We discuss user feedback in terms of the observations and results of a short game-play study based on a prototype of the card game.

Fig. 1 illustrates the game constellation of REFACTORY. A game board represents a software system consisting of multiple sub-systems (each comprising multiple software components in turn), which are assigned to players at start (see ① in Figure 1). Goal of the game is basically to increase the value of the given software system by realizing new functionalities assigned every sprint [36] (see ②). However, the players are impeded in realizing new functionalities by pre-existing issues in the software design (see ①), which complicate the modification and extension of components and thereby increase development time. These bad smells can be removed by applying certain combinations of refactoring techniques (see ③). In addition, events (such as code reviews or a feature check from the customer) impact the value scores (see ②a). Further details on the design and game-play workflow can be found in Section 3.3.

The remainder of this paper is structured as follows. Section 2 discusses game approaches related to the proposed card game. In Section 3, our game REFACTORY is explained in detail, including the targeted objectives (e.g. competences and learning context), the game design (e.g. concepts and mechanics) as well as the workflow and variants of game play. Then, we report first user feedback in terms of the results of a small game-play study (see Section 4). Section 5 reflects the approach's limitations and further potential. Section 6 concludes the paper.

RELATED WORK

In recent years, several (educational) games have been proposed in the field of software engineering such as approaches for game-based learning, serious gaming and gamification; see e.g. [6, 32, 21, 26, 2]. Among these gaming approaches, closely related to our approach are in particular games for software refactoring on the one hand and educational card games on the other hand, which are both discussed in the following.

Games for Software Refactoring

For the sub-field of software refactoring, only very few approaches of (educational) games can be identified. Based on the systematic literature reviews [26, 32, 2] and further research, the following gaming approaches related to refactoring can be identified [34, 33, 8, 20, 16, 3]. For an analysis of refactoring games, also see [15]. Among these, serious games and gamified (editor-based) development environments such as [8, 16, 3] provide realistic or real-world development conditions enhanced by gaming elements to motivate the software developers. For instance, the serious game proposed in [16] supports several single and multi-player game modes striving

to increasing a system's internal quality by reducing its technical debt via refactoring. For this purpose, analysis tools such as test frameworks and technical-debt analyzers are integrated. *CodeArena* [3] is a 3D game built on *Minecraft Forge* that allows for fighting code clones (represented as monsters) via code refactoring based on an additional editor view. Related to these are intelligent editor-based tutoring systems such as [35, 17]. They all have in common that they require a high level of entry skills, e.g. regarding programming, software-design principles or handling of the programming environment (e.g. the test framework). In addition to these approaches, only very few game-based learning approaches are established such as [34, 33], which provides an interactive learning path with several activities and learning units (e.g. multiple-choice questions) based on tangible cards representing smell types and an interactive screen for visualizing refactoring options for selected smell types in terms of a dependency graph. Complementing these games, we propose a card game for actively learning certain principles of software refactoring (e.g. combining refactoring techniques to remove smells, strategical aspects on balancing costs and value of refactoring). Our game has low entry barriers and can be combined with these more demanding games.

Card Games for Software Engineering

In recent years, several card games have been proposed in the field of software engineering; see e.g. [4, 27, 23, 11, 1, 25, 7]. For instance, *planning poker* is a popular gamified method for a group-based estimation of development-task effort, widely used in agile software projects, see e.g. [27]. An example for teaching programming-related concepts is *Potato Pirates* [1], which aims at mediating fundamentals in computational thinking (especially to a younger audience) in terms of basic programming structures (conditionals, loops, variables etc.). More related to our approach are card games that simulate the software development process, such as *Mission to Mars* [23] for agile release planning, or *Problems and Programmers* according to phases of the waterfall process model [4]. Moreover, *DecidArch* allows for training to handle design decisions in software architecture [25, 7]. The game also includes cards representing events that require the players to react by meaningfully applying design decisions. As a complement to these card games, our game focuses on handling the process of software refactoring. To the best of our knowledge, the *Dice of Debt Game* is the only card game related to (learning) software refactoring [11]. *Dice of Debt* simulates an agile software project confronted with technical debt. The debt can be reduced by applying certain (cost intense) measures represented in terms of cards for the techniques of *continuous integration*, *reducing complexity*, *increased test coverage*, and *code review*. For monitoring the progress, it provides a scoring and tracking sheet. To this extent, the game has a few aspects similar to ours such as the aim to increase a quality score (reducing technical debt) by applying certain counter measures. However, in contrast to *Dice of Debt*, which focuses on the high-level management of technical debt, our game addresses, among other aspects, on the combinations of concrete refactoring techniques for removing types of bad smells (i.e. debt items). Moreover, *Dice of Debt* includes a high factor of chance by quantifying the cost and value of debt counter-measures with multiple dices, while our game more focuses on strategic aspects of balancing costs and benefits of refactoring in order to raise awareness for efficiently and meaningfully applying refactoring techniques.

REFACTORY

In this section, we introduce our card game REFACTORY. In particular, we describe the game's objectives, detail the game design in terms of the basic game concepts and game mechanics and, finally, describe the game-play workflow and exemplary game variants.

Objectives

Objective of REFACTORY is (1) to motivate players for learning more about software refactoring and (2) to mediate basic competences, which are explained below.

Motivation

The game aims to motivate players by providing a playful simulation of the challenges in handling technical debt in software systems expressed as concrete bad code smells. The aesthetic aspects [19] addressed by the game are *narrative* (role as software developers), *challenge* (removing smells and increasing the software's value) and also *fellowship* (competing and collaborating players in a social framework). From the perspective of gamer types [5], the card game addresses primarily *killers* that aim to rank and to win (score sheets) and *achievers* that aim to achieve a status or goals (e.g. by removing all smells). The group-oriented variants also address *socializers*. Moreover, the game provides balance between elements of chance (e.g. dicing for assigning smells and functionalities or selecting random event cards), strategy (e.g. deciding how to invest time points) and knowledge application (e.g, combining refactoring techniques to remove smells).

Competences

In addition to fostering motivation, the game also aims at mediating principles in software refactoring, which can be described in terms of target competences (a.k.a. learning objectives). For specifying competences (especially in engineering and computing), Bloom's revised taxonomy of educational objectives [22] is very popular. Moreover, Paquette distinguishes between *prerequisite* and *target competences* [31]. Built on these, we have developed a framework for refactoring competences [13] that is used below. In particular, the game already requires a minimal degree of *understanding* basic terms of refactoring (*factual* knowledge; i.e. I, 2, A/B according to [13]). In turn, the game mainly addresses the following two *target competences*:

- At first, the game addresses the application of a combination of refactoring techniques to remove certain smells, which can be classified as processing *conceptual* knowledge at the cognitive levels of *application*, *analysis*, and *evaluation* (i.e. II, 3/4/5, B).
- Moreover, it also addresses *applying* and *analyzing metacognitive* knowledge aspects in terms of strategies for refactoring expressed as balancing the time/costs and benefits of performing refactoring (i.e. IV, 3/4, B).

Learning Context

The game can be seen as a complement to other learning and training activities and environments (didactic mix), which are discussed in the section on related work (see above). The game can be described as a *concepts-first* approach to introducing software refactoring. It addresses students or even practitioners (related to software projects) that are, however, novices in the field of software refactoring. In particular, the game especially focuses on players without programming-related experiences or that are more oriented to project management. For example, the game could be used in the framework of a university course on software design, maintenance or agile development. Based on the addressed competences specified above, a competence-oriented learning path can be described, in which the game represents a starting point guiding. Learners can be guided from there to higher levels of competence, e.g. via tutoring systems [17] and serious games [16].

Development Process

The development of the game is oriented to a process model, which includes the use of a domain ontology for games in software refactoring. Fig. 2 depicts the model in terms of a UML activity diagram representing the key development steps, which are elaborated in the further Sections.

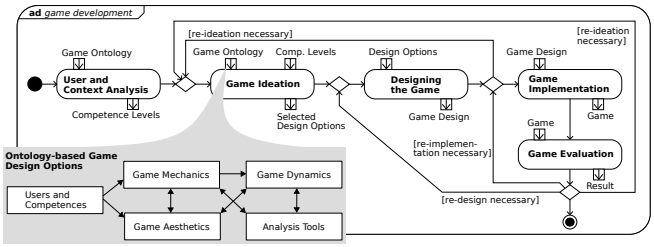

Figure 2. UML activity diagram representing the applied process for developing the card game [14].

Further details on the ontology-based analysis and design of games and the included process model can be found in [14].

Game Design

In the following, we give a conceptual overview of the game and explain the basic game mechanics and components.

Conceptual Overview

Fig. 3 shows an overview of concepts applied in REFACTORY in terms of a class diagram of the Unified Modeling Language (UML2) [29]. The concept map represents the key elements of the game and the relationships between them.

The `Game Play` is based on a `Game Board` representing a software system consisting of multiple `Sub-Systems` (including multiple `Software Components` in turn; see Fig. 3. Each subsystem is assigned to a `Player` (in the role of a software developer). In a `Game Play`, multiple `Players` strive for increasing the `Value` of the software system. A game play consists of multiple `Rounds` representing agile sprints, during which functionalities are assigned to be realized, smells can be removed by applying refactoring and additionally events occur, which can impact each players' value. These aspects are realized by the following four kinds of `Cards` (see Fig. 4).

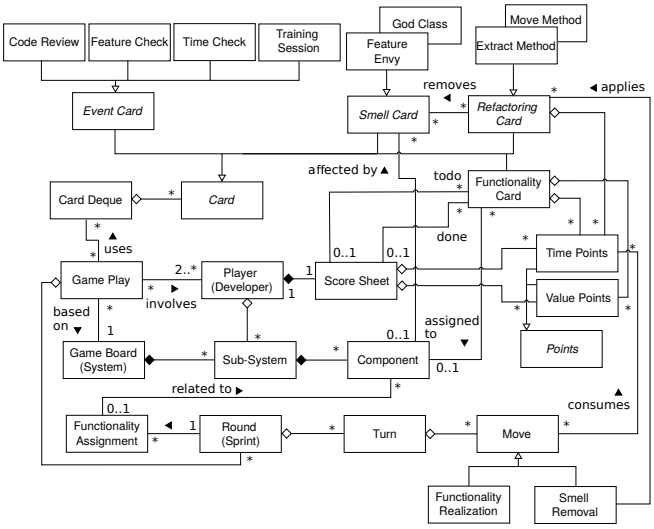

Figure 3. Conceptual overview of card-game elements.

(a) `Smell Card`: representing a *bad code smell* that negatively impacts the maintainability and extensibility of the affected software component (see e.g. [9, 37]). A smell card provides a textual description of the nature of the smell (front of the card) and a list of combinations of refactoring techniques to remove the smells (on the back of the card; see (a) in Fig. 4). At game start, the components are already affected by bad smell cards (diced), for which the corresponding cards are located at the components of the game board.

(b) `Refactoring Card`: representing a concrete refactoring technique, such as MOVEMETHOD or EXTRACTCLASS (see e.g. [9]) that aims at improving the internal quality by modifying a code fragment while preserving the observable software behavior [30]. A refactoring card provides information on the technique and the effort to perform it in terms of `Time` points (see (b) in Fig. 4).

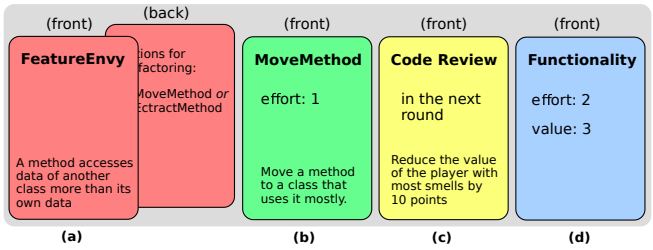

Figure 4. Exemplary cards of (a) bad smells, (b) refactoring techniques, (c) events, and (d) functionalities.

(c) `Event Card`: representing events that impact the software project and the strategies of balancing the removement of bad smells and realizing functionalities (see (d) in Fig. 4). In particular, the following four kinds of events are distinguished:

- `Code Review`: representing a review of the code quality with the consequences of reducing the value points of the player with the most smells.

- `Feature Check`: representing a check of the current state of functionality realization from the customer (or product owner) with the consequence of reducing value points of the players with the most functionalities *todo* (i.e. not yet realized).
- `Time Check`: representing a check of the time not used by the players with the consequence of reducing the value points of the player with the most available (i.e. unused) time points.
- `Training Session`: representing a session dedicated to training techniques for software refactoring resulting in additional refactoring cards drawn by the players. The training can be applied to all players or just the player who draws the card.

The events take place immediately or in a future round, as noted on the front of the card (see (c) in Fig. 4).

(d) `Functionality Card`: representing a certain functionality of the software system. The details of the functionality are hided in the game, i.e. that only the effort (in terms of time points) to realize the functionality (*todo*) and the value (also in terms of points) of the functionality (in case it has been realized; see below) are noted on the front of the card (see (d) in Fig. 4).

In every `Round`, `Time` points are given to each player, in terms that they are noted into a player's `Score Sheet`. The sheet also gives an overview of the player's functionalities *todo* (for which the target components are noted) and *done* as well as the achieved `Value` (as the sum of functionalities done). Moreover, a set of new `Functionality Cards` (*todo*; taken from the product backlog) are drawn from the card deck and assigned to the players' components by dicing. Each round, one player (rotating) also selects an `Event Card` that impacts all players or only the acting player (see above). Per round, each player then moves in turn. At first, she draws new `Refactoring Cards`. `Functionality` and `Refactoring Cards` are then taken in player's hand. Each player then basically can decide how to invest the `Time` points for performing the following two kinds of `Moves`.

- `Functionality Realiziation` represents a player's move to realize a functionality, which is at the player's hand (*todo*; previously drawn). By realizing a functionality, the player's time points are reduced by the corresponding effort points and in turn the player's value points are increased by the functionalities' value (in the score sheet). The functionality card is then placed at the corresponding component on the game board. In case a component is affected by a bad smell, the costs of realizing the functionality are doubled. While realizing a functionality, new smells can be introduced potentially (by dicing a certain number). For this purpose, then the corresponding cards are drawn from the card deck and located at the corresponding component.
- `Smell Removal` represents the combination of refactoring techniques to remove a certain bad smell located at a player's component. For this purpose, (1) the effort points of the refactorings must be available (and are consumed) and (2) the combination must be appropriate for removing the smell (see Table 1).

Further details on game play are explained in the following Section.

Mapping Refactorings to Bad Smells

Table 1 presents exemplary applied mappings between bad code smells and corresponding refactoring techniques, i.e. options and combinations to remove the smells. The mappings are based on the documented knowledge provided by [9, 37].

Table 1. Exemplary mappings between bad smells and refactoring techniques according to [9, 37].

Bad Smell [9]	Refactoring techniques
`FeatureEnvy`	`MoveMethod` or `ExtractMethod`
`MessageChain`	`MoveMethod` and `ExtractMethod`
`SwitchStatements`	`MoveMethod` and `ExtractMethod`
`GodClass`	`ExtractClass` or `MoveMethod` or `ExtractMethod`
`LongMethod`	`ExtractMethod`
`DataClump`	`ExtractClass`
`ShotgunSurgery`	`MoveMethod`

Game Play

Based on the game design explained above, game-play aspects such as how to prepare a game session as well as scenarios and variants of game play are described below.

Game Preparation

For playing the game, a game board, score sheets and game cards [1] as well as a traditional dice (i.e. faces one to six) and a pen (to fill the score sheet) are required. Before game play, the subsystems (with included components) have to be assigned to players. Moreover, for each subsystem, smell cards are allocated to components by random (diced) and each player also receives a starter set of refactoring cards (see ① in Fig.1).

Game-Play Scenarios

As shown above, the game provides strategic challenges, demands for applying conceptual knowledge and contains a certain degree of chance (e.g. dicing). In the following, exemplary game-play aspects are detailed in terms of scenarios. For instance, for managing the technical debt items (concretely expressed as bad smells) located in their sub-systems, the players can select between different strategies that are very close to the options actually used in refactoring practice.

- At first, a player can realize a functionality in a smelly component by ignoring the bad smell, which can be motivated by different reasons, such as inexperience or unawareness on the one hand, or driven by the assumption that this component won't be often modified. However, every modification of a smelly components can become quite cost expensive (cf. *technical-debt interest* [24]).
- A second option for the player is to remove the smells just before realizing funtionalities, which is referred as *ad-hoc* or *floss* refactoring [28]. The advantage of this technique is that only smells are removed that are actively maintained and/or extended.
- The third option is to remove (all) smells in anticipation, even if no concrete extension by functionalities is planned. In this strategy, the player/developer runs the risk to prevent

[1] The materials of the card-game prototype (e.g. game board, score sheet, game cards) are available for download from http://refactoringgames.com/cardgame/.

issues in terms of debt interest that will never accrue, since the affected components will eventually never be touched by developer.

Game-Play Variants

The game design and game constellation specified above allow for describing multiple game variants, which differ regarding the user interaction and at the level of detail in the rules of the game. Three exemplary variants are shortly described below.

(A) *Competing Players*: The players compete against each other in collecting value points. Each player is only responsible for their subsystems. This variant represents the base for the following two game variants.

(B) *Competing Groups*: In this variant, multiple groups compete against each other. In contrast to (A), the members of a group support each other in several regards, such as for combining refactoring cards or realizing functionalities of each other.

(C) *Collaborating Players*: The third variant describes a team of all participating players that aims at optimizing the value of the software system. In this, the players can support each other in exchanging refactoring cards and balancing the players' time for realizing the assigned features. A possible reference for comparing the team's value could be the value achieved in previous sessions (when applying other strategies) or achieved by other teams.

USER FEEDBACK

We investigated user feedback on the game via a two-stage process (1) by performing a pilot to identify obvious issues and to refine the game and (2) via a user study with 19 participants, which are both explained in the following.

Pilot

In order to test the game functionality of and gain first feedback on game play, we conducted a short pilot with three software developers playing a first version of the game. We observed the game play and discussed the experiences with the players afterwards. The players gave generally positive feedback and confirmed simplicity as key for learning principles in software refactoring. In addition, a few suggestions to improve the game have been collected, such as adding more elements of chance to increase game fun and to improve the game theme to increase the acceptance, especially by a younger audience.

Game-Play Study

Based on improvements of the game design (such as introducing event cards; see above), we conducted an additional small user study with 19 voluntary participants, consisting of 7 software developers and 12 students of the *Information Systems* bachelor studies, of which 8 indicated to have at least some experiences in software engineering, while 4 had little/none; all with low to medium knowledge in software refactoring. The players were instructed into the gaming rules before game play. Game sessions (of each 8 rounds) have been performed in five groups, each consisting of 3 to 4 players, which took about 20 to 40 minutes to complete. Every group has explored two game variants, i.e. at first (A) *competing players* and then one of the group-oriented variants (B by two groups, C by three groups). After that, the participants have been asked to specify the level of agreement to 8 given statements (see Table 2) via *Likert* scales (1=no to 6=full agreement), which can be structured by the following 3 blocks.

(1) general impressions of the game,
(2) fun and motivation of game play (*first objective*), and
(3) the game as mediator of competences (*second objective*)

Table 2. Statements on game play (ordered by blocks 1 to 3).

No.	Statement
1.1	By playing the game, I learned more about software refactoring.
1.2	By playing the game, I have a better understanding of the value of software refactoring.
1.3	The rules of the card game were easy to follow.
2.1	The card game was fun to play.
2.2	The card game motivated me to learn more about software refactoring.
2.3	Playing in a group was more motivating for me than playing as a single player.
3.1	By playing the game, I learned more on how to combine refactoring techniques to remove bad smells.
3.2	By playing the game, I reflected on balancing costs and benefits of applying refactoring.

General Impressions

Fig. 5 shows the results on statements regarding the general impression of the game. Almost all participants (i.e. 16) agreed to the statement that the game supports in learning principles in software refactoring (1.1). In addition, most (63%) indicated that playing the game helps better understanding the value of refactoring (1.2). Interestingly, many participants (i.e. 13) have experienced problems in following the game rules (1.3), which can be induced by the explanation of the rules or by the complexity of the rules themselves (also see the discussion in the following section).

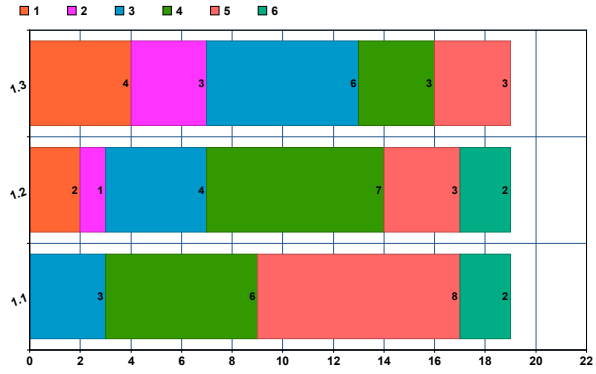

Figure 5. (1) General impressions of the game.

Fun and Motivation

Fig. 6 reports on the answers to statements regarding fun and motivation in playing the game (*first objective*). In general, 63% of the participants indicated that playing the game is fun (2.1). The statement that the game motivates to learn more about software refactoring (2.2) finds overall weak support. Moreover, almost all participants stated that the group-oriented variants (i.e. competing groups and collaborating players) were more fun to play (2.3).

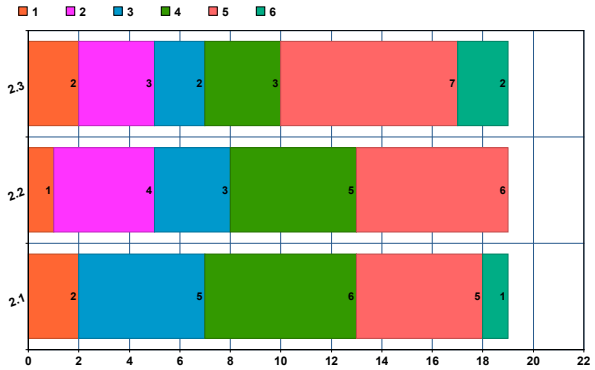

Figure 6. (2) Fun and motivation of game play (*first objective*).

Competence Acquisition

Fig. 7 depicts the results on statements regarding competence acquisition (*second objective*). Most participants (i.e. 79%) indicate that the game can support acquiring competences for software refactoring. In detail, most participants agreed on the statement that the game supports improving the knowledge on how to combine refactoring techniques to remove bad smells (3.1). Moreover, almost all participants (i.e. 15 out of 19) agreed that by playing the game they reflected on balancing costs and benefits of refactoring (3.2).

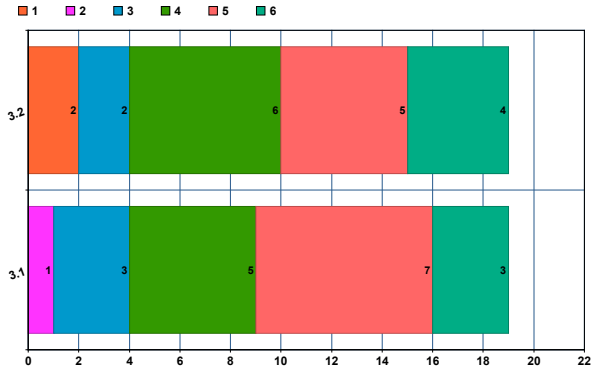

Figure 7. (3) The game as mediator of competences (*second objective*).

Improvement Potential

In addition to these *closed questions*, the participants were also asked for potential game improvements, which mainly dispersed over the game's appearance/theme and the gameplay rules. Some participants have indicated that the game could be visually and haptically more appealing and that the theme of the game needs to be revised. Moreover, some noticed that the game is a bit too complicated and –correlating to this– that the manual should be revised. Among other aspects, these points are discussed in the following section.

DISCUSSION

The short game-play study indicates that the developed card game supports in acquiring basic competences for software refactoring and also motivates the participants to deal with software refactoring. In the following we reflect on limitations regarding the user study and the game approach as well as on potential for further improvement. At first, the study is based on a small and heterogeneous target group consisting of developers and IT students differing in terms of knowledge and experience, who can be seen as typical card-game players. However, a potential threat to the *internal validity* can be seen in the kind of questions asked to the participants in the sense that the intention behind the study is transparent to the participants, i.e. they can probably recognize that their agreement with the approach presented is desired. Thus, the participants can be seen as biased to some extent. To overcome these limitations, further experimentation and refinement is required, which is planned in future research. In particular, it is intended to perform further user studies with more participants in the framework of university courses and by measuring the competence acquisition via exemplary tasks that relate to competence levels [13]. In this course, it would also be interesting to investigate how the game actually addresses player types (such as achievers or killers) [5].

In general, the proposed card game focuses on certain conceptual competences (see above), which can be seen as a starting point to learning software refactoring. For the player, this means that it remains to be explored, for example, how to actually perform an EXTRACTCLASS by performing code modifications (in a larger code base). However, we believe that the card game can also foster other skills, such as problem-solving, team work, or communication (e.g. resulting from face-to-face interaction and immediate social feedback), which could be investigated in further research.

The realized non-digital card game was motivated by the benefits of lower entry barriers and the potential to focus on selected conceptual aspects of software refactoring. However, the proposed game design could also be realized as a digital game, which would come with some advantages, such as an automated assessment and game processing. In addition, there is certainly a lot of potential in improving the visual appearance of the game, which could correlate with a refinement of the game mission. For instance, within a *battle* narrative, the bad smells could by represented as *enemies* that can be beaten by deftly applying refactoring *weapons*. Building upon this, the cards could be visually improved, e.g. oriented to the popular fantasy card game *Magic: The Gathering* [10].

CONCLUSION

In this paper, we have introduced REFACTORY, a non-digital card game for teaching basic concepts in software refactoring. The short user study indicated that the card game can (1) motivate for learning more about software refactoring and (2) support learning refactoring principles (i.e. combining refactoring techniques to remove bad smells and of applying and reflecting strategies for balancing costs and benefits). For future work, we intend to apply the game within university courses and to perform further user studies. In order to improve the user experience, we also plan to revise the game layout (as discussed above). Another possible direction could be to transform the presented non-digital into a digital game, by which the assessment of actual code modifications could be included (integrated editor) and which would also allow a combination and integration with other training/gaming activities and environments (see e.g. [16, 17]).

Acknowledgements

This work was partly funded by the Department for Economic Affairs, Labour and Statistics (MA23) of the City of Vienna (Austria) through the research project "New Work – New Business" at the UAS BFI Vienna.

REFERENCES

1. Lim Jia Xuan Aditya Batura, Fendy Lieanata and Seah Tat Leong. 2017. Potato Pirates. (2017). Codomo Pte Ltd. `https://www.potatopirates.game` [September 6, 2019].
2. Manal M Alhammad and Ana M Moreno. 2018. Gamification in software engineering education: A systematic mapping. *J. Systems and Software* 141 (2018), 131–150.
3. Simon Baars and Sander Meester. 2019. CodeArena: Inspecting and Improving Code Quality Metrics in Java using Minecraft. In *Proceedings of the 2019 International Conference on Technical Debt (Tool Demos)*. IEEE.
4. Alex Baker, Emily Oh Navarro, and André van der Hoek. 2005. An experimental card game for teaching software engineering processes. *Journal of Systems and Software* 75, 1-2 (2005), 2–16.
5. Richard Bartle. 1996. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD research* 1, 1 (1996), 19.
6. Craig Caulfield, Jianhong Cecilia Xia, David Veal, and S Maj. 2011. A systematic survey of games used for software engineering education. *Modern Applied Science* 5, 6 (2011), 28–43.
7. Remco C de Boer, Patricia Lago, Roberto Verdecchia, and Philippe Kruchten. 2019. DecidArch v2: An improved Game to teach Architecture Design Decision Making. (2019), 153–157.
8. Leonard Elezi, Sara Sali, Serge Demeyer, Alessandro Murgia, and Javier Pérez. 2016. A game of refactoring: Studying the impact of gamification in software refactoring. In *Proc. of the Scientific Workshops of XP2016*. ACM, 23.
9. Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. 1999. *Refactoring: improving the design of existing code*. Addison-Wesley Professional.
10. Richard Garfield. 1993. Magic: The Gathering. *Board Game. Wizards of the Coast, Renton, Washington, US* (1993).
11. Tom Grant. 2015. Dice of Debt Game. (2015). GameChange LLC. `https://www.agilealliance.org/dice-of-debt-game/` [September 6, 2019].
12. Thorsten Haendler and Josef Frysak. 2018. Deconstructing the Refactoring Process from a Problem-Solving and Decision-Making Perspective. In *Proc. of the 13th International Conference on Software Technologies (ICSOFT)*. SciTePress, 363–372.
13. Thorsten Haendler and Gustaf Neumann. 2019a. A Framework for the Assessment and Training of Software Refactoring Competences. In *Proc. of 11th International Conference on Knowledge Management and Information Systems (KMIS)*. SciTePress.
14. Thorsten Haendler and Gustaf Neumann. 2019b. Ontology-based Analysis and Design of Educational Games for Software Refactoring. In *Computers Supported Education, Revised Selected Papers of CSEDU 2019*. Springer.
15. Thorsten Haendler and Gustaf Neumann. 2019c. Ontology-based Analysis of Game Designs for Software Refactoring. In *Proc. of the 11th International Conference on Computer Supported Education (CSEDU)*, Vol. 1. SciTePress, 24–35.
16. Thorsten Haendler and Gustaf Neumann. 2019d. Serious Refactoring Games. In *Proc. of the 52nd Hawaii International Conference on System Sciences (HICSS)*. 7691–7700.
17. Thorsten Haendler, Gustaf Neumann, and Fiodor Smirnov. 2019. An Interactive Tutoring System for Training Software Refactoring. In *Proc. of the 11th International Conference on Computer Supported Education (CSEDU)*, Vol. 2. SciTePress, 177–188.
18. Juho Hamari, Jonna Koivisto, and Harri Sarsa. 2014. Does gamification work?–a literature review of empirical studies on gamification. In *Proc. of 47th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 3025–3034.
19. Robin Hunicke, Marc LeBlanc, and Robert Zubek. 2004. MDA: A formal approach to game design and game research. In *Proc. of the AAAI Workshop on Challenges in Game AI*, Vol. 4. AAAI Press San Jose, CA, 1–5.
20. Shivam Khandelwal, Sai Krishna Sripada, and Y Raghu Reddy. 2017. Impact of Gamification on Code review process: An Experimental Study. In *Proc. of the 10th Innovations in Software Engineering Conference*. ACM, 122–126.
21. Mehmet Kosa, Murat Yilmaz, Rory O'Connor, and Paul Clarke. 2016. Software engineering education and games: a systematic literature review. *Journal of Universal Computer Science* 22, 12 (2016), 1558–1574.
22. David R Krathwohl. 2002. A revision of Bloom's taxonomy: An overview. *Theory into practice* 41, 4 (2002), 212–218.
23. Philippe Kruchten and James King. 2011. Mission to Mars: An agile release planning game. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 552–552.
24. Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. 2012. Technical debt: From metaphor to theory and practice. *IEEE software* 29, 6 (2012), 18–21.
25. Patricia Lago, Jia F Cai, Remco C de Boer, Philippe Kruchten, and Roberto Verdecchia. 2019. DecidArch: Playing Cards as Software Architects. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*.
26. Michael A Miljanovic and Jeremy S Bradbury. 2018. A Review of Serious Games for Programming. In *Joint International Conference on Serious Games*. Springer, 204–216.
27. Kjetil Moløkken-Østvold, Nils Christian Haugen, and Hans Christian Benestad. 2008. Using planning poker for combining expert estimates in software projects. *Journal of Systems and Software* 81, 12 (2008), 2106–2117.
28. Emerson Murphy-Hill, Chris Parnin, and Andrew P Black. 2012. How we refactor, and how we know it. *IEEE Transactions on Software Engineering* 38, 1 (2012), 5–18.
29. Object Management Group. 2017. Unified Modeling Language (UML), Superstructure, Version 2.5.1. (2017). `http://www.omg.org/spec/UML/2.5.1` [September 6, 2019].
30. William F Opdyke. 1992. *Refactoring object-oriented frameworks*. University of Illinois at Urbana-Champaign Champaign, IL, USA.
31. Gilbert Paquette. 2007. An ontology and a software framework for competency modeling and management. *Educational Technology & Society* 10, 3 (2007), 1–21.
32. Oscar Pedreira, Félix García, Nieves Brisaboa, and Mario Piattini. 2015. Gamification in software engineering–A systematic mapping. *Information and Software Technology* 57 (2015), 157–168.
33. Felix Raab. 2012. CodeSmellExplorer: Tangible exploration of code smells and refactorings. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*. IEEE, 261–262.
34. Felix Raab, Markus Fuchs, and Christian Wolff. 2012. CodingDojo: Interactive slides with real-time feedback. *Mensch & Computer 2012–Workshopband: interaktiv informiert–allgegenwärtig und allumfassend!?* (2012).
35. Mincho Sandalski, Asya Stoyanova-Doycheva, Ivan Popchev, and Stanimir Stoyanov. 2011. Development of a Refactoring Learning Environment. *Cybernetics and Information Technologies (CIT)* 11, 2 (2011).
36. Ken Schwaber and Mike Beedle. 2002. *Agile software development with Scrum*. Vol. 1. Prentice Hall Upper Saddle River.
37. Girish Suryanarayana, Ganesh Samarthyam, and Tushar Sharma. 2014. *Refactoring for software design smells: Managing technical debt*. Morgan Kaufmann.
38. Ewan Tempero, Tony Gorschek, and Lefteris Angelis. 2017. Barriers to refactoring. *Commun. ACM* 60, 10 (2017), 54–61.