Gamification: a Game Changer for Managing Technical Debt? A Design Study

Matthieu Foucault, Margaret-Anne Storey
University of Victoria
The CHISEL Group
Victoria, BC, Canada
Email: mfoucault@uvic.ca, mstorey@uvic.ca

Xavier Blanc, Jean-Rémy Falleri
Université de Bordeaux
LaBRI – Software Engineering Group
Bordeaux, France
Email: xblanc@labri.fr, falleri@labri.fr

Cédric Teyton
ProMyze
Bordeaux, France
Email: cedric.teyton@promyze.com

Abstract—Context: Technical debt management is challenging for software engineers due to poor tool support and a lack of knowledge on how to prioritize technical debt repayment and prevention activities. Furthermore, when there is a large backlog of debt, developers often lack the motivation to address it. Objective: In this paper, we describe a design study to investigate how gamification can support Technical Debt Management in a large legacy software system of an industrial company. Our study leads to a novel tool (named Themis) that combines technical debt support, version control, and gamification features. In addition to gamification features, Themis provides suggestions for developers on where to focus their effort, and visualizations for managers to track technical debt activities. Method: We describe how Themis was refined and validated in an iterative deployment with the company, finally conducting a qualitative study to investigate how the features of Themis affect technical debt management behavior. We consider the impact on both developers and managers. Results: Our results show that it achieves increased developer motivation, and supports managers in monitoring and influencing developer behaviors. We show how our findings may be transferable to other contexts by proposing guidelines on how to apply gamification. Conclusions: With this case, gamification appears as a promising solution to help technical debt management, although it needs to be carefully designed and implemented to avoid its possible negative effects.

I. INTRODUCTION

The technical debt (TD) metaphor was first used in 1992 to refer to source code quality issues: “Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite.” [8] The debt analogy remains relevant more than 25 years later as development cost is one of the main issues facing today’s software engineering practitioners [23]. Although the TD metaphor applies to several aspects of software developments, including requirements, architecture, and documentation, TD in source code remains the most widely studied [26]. There are various ways to manage technical debt in code, from repayment of TD by rewriting code to make it more maintainable, to preventing TD from occurring in the first place, to monitoring TD to bring awareness of how it changes over time [26].

Technical debt management (TDM) is difficult for software engineers due to poor tool support [1] and a lack of knowledge on how to prioritize TD repayment and prevention activities. But more importantly, it is challenging because many developers lack the mindset and motivation to focus on avoiding or repaying technical debt—many developers consider TD repayment as a time consuming activity that is not guaranteed to provide immediate benefits [43].

TDM can be even more challenging if the project’s code base is large and contains a backlog of technical debt from many years of development: our research group was contacted by an industry partner (the French governmental agency Pôle Emploi) when they realized their 10-year-old project with more than 550k lines of code contained an extensive backlog of code TD that would require over 2,000 worker-days to fix. This code TD comes in the form of poor code constructs (also named code smells [3]) that eventually cause a loss of quality and productivity [30]. Our industry partner recognized that repaying the TD is essential, but would eventually lead to countless repetitive development tasks without adding any visible value from the developer’s perspective. Similarly, they realized that developers lacked the motivation to avoid adding new TD because adding new code smells would have little impact on the already high load of existing debt. Thus their problem was twofold: they needed to motivate developers to both repay existing and avoid adding new TD, and they needed a mechanism that managers could use to track and influence how developers acted towards TDM.

In this paper, we present a problem-driven design study [34] to address the challenge of managing TD. Our proposed solution to this challenge involved the introduction of gamification to address TD. By gamification, we refer to the use of game elements in a non-game context [12]. We chose gamification as it has already been used in software engineering and shows some promise for improving software processes [7], [15], [5], although the empirical evidence of its impact on developer behaviour and motivation is scarce.

Working closely with our industry partner and using a design study methodology (see Section III), we characterized the problem we aimed to address through our research. Then we designed and customized a gamification tool, Themis, that integrates with our partner’s project version control tool as...
well as with the SonarQube tool for identifying and measuring TD (see Section IV). Themis uses gamified features such as points, leaderboards, and challenges as a way to motivate developers and help managers with TDM. The tool was designed in an iterative manner (through an early deployment of the tool) in response to the needs elicited from our partner.

We also studied how Themis influences developer and manager behaviours and increases motivation towards managing technical debt by surveying both managers and developers after they used it for three months (see Section V). We discovered how and why Themis and, in turn, gamification positively influences developer behaviour and motivation as well as how it supports managers. But we also learned that gamification must be treated with care as it may not be suitable for all developers and project contexts.

To motivate the need for this research, Section II presents some background on technical debt, describes tools for detecting technical debt, and provides an overview of gamification concepts and how gamification has been applied in software engineering. In Section VI we consider how our findings may be transferable to other developer contexts and suggest guidelines for how gamification may be applied to the task of technical debt management. We conclude the paper by making a plea for more research on this topic and present some future research directions (see Section VII).

II. BACKGROUND

We provide background and discuss related work on technical debt and tools for measuring it. We also introduce the concept of gamification and review how it has been applied in software engineering.

A. Technical Debt

Although the technical debt (TD) metaphor has been used for many years, the formalization of TD is still a work in progress [37] and most research efforts studying TD are fairly recent. In Li et al.’s systematic mapping study on technical debt, only four primary studies on the topic of TD were identified between 1992 and 2008, while at least 15 studies have been published each year since 2010 [26] and a special issue on technical debt was published in the Journal of Systems and Software in 2016 [10].

As with most metaphors, TD is subject to interpretation and its definition can vary. TD can be related to a wide range of software artifacts including source code, requirements, documentation, development process, architecture, and even people (e.g., having software expertise concentrated in too few people) [8, 26]. TD may be created deliberately or inadvertently and it can be reckless or prudent [17].

In this paper, we consider code TD, whether it is deliberately or inadvertently added. To help managing code TD, developers use tools called linters to automatically identify code smells [4] and pinpoint parts of the code that should be fixed to repay the debt [9, 25]. The code smells considered by these linters are described by rules that can be automatically checked against a given source code thanks to static analysis. Some examples of linters are PMD [3] or Checkstyle [4] for Java and ESLint [4] for JavaScript. For example, in ESLint, the no-unreachable rule detects code that is unreachable (after return, throw, break, and continue statements) Listing 1 presents a snippet of JavaScript source code highlighting this rule.

```javascript
function foo() {
    return true;
    console.log("done");
}

function bar() {
    throw new Error("Oops!");
    console.log("done");
}

while(value) {
    break;
    console.log("done");
}
```

Listing 1. Example of invalid code for the no-unreachable rule.

Li et al.’s study [26] identified that technical debt management consists of different kinds of activities: TD identification, TD measurement, TD prioritization, TD monitoring, TD repayment, TD representation/documentation, TD communication, and TD prevention. Their mapping study showed that the most investigated activities are TD identification (code analysis, dependency analysis) and TD measurement (calculation models, code metrics). In practice, TD communication is the most commonly used activity as reported by Yli-Huumo et al. [43]. However, they stated that “the biggest issue with TD communication has been the gap between technical and non-technical stakeholders”, which emphasizes the need for tools that help teams communicate about the state of a project’s technical debt. Yli-Huumo et al. also found that providing developers with the proper mindset and motivation for TDM is one of the largest challenges as technical debt management and repayment takes time and can be seen as a waste of effort by stakeholders [43].

B. Gamification

Gamification is mostly defined as the use of game design elements in non-game contexts [12]. This definition implies that a gamified application: 1) is a game and has rules defining (at least) player interactions and quantifiable outcomes [21]; 2) uses game elements such as feedback, reputation, and rank [32]; and 3) has a game design that may make use of challenges, time pressure, or levels [6].

Gamification is a young domain where few theoretical foundations are available [33]. The emerging theories focus on player motivation, behavior change and engagement, with specific attention paid to the relationship between intrinsic motivation (aligned with the player’s inner values) and extrinsic motivation (coming from external factors) [11]. The objective of gamification is to increase the intrinsic motivation based on extrinsic motivators. For example, the desire to become

http://pmd.sourceforge.net/pmd-4.3.0/rules/index.html
http://checkstyle.sourceforge.net/checks.html
http://eslint.org/
a better programmer (an intrinsic motivation) may be realized and enhanced through gaining badges (an extrinsic motivator). However, care must be taken to ensure that extrinsic motivators do not lead to decreased intrinsic motivation [11].

Research on the use of gamification in software engineering is relatively recent and most studies have focused on the design of tools that introduce gamification [29]. Sheth et al. proposed a framework called HALO to add gamification into a software engineering environment with the objective to enhance productivity [36]. This framework was later used to improve the teaching of software design and testing [35]. Singer and Schneider developed a system using points, badges, and leaderboards to provide an incentive for developers to commit their code more often. After conducting an experiment with 37 students, their interviews showed that the tool increased the participants’ awareness of the other developers’ activity [39]. Steffens et al. developed a preliminary framework of how gamification can be used to support and improve collaboration in software engineering [40], as did Dal Sasso et al., who also proposed a framework to create gamified environments in software engineering [10]. Vasilescu et al. investigated how gamification elements (such as reputation points and badges) enhance social knowledge sharing on and across sites such as GitHub and Stack Overflow [22]. Passos et al. explored how different releases of a product can be mapped to gaming levels [28]. Snipes et al. proposed how gamification can be used to improve developers’ coding activities (e.g., refactoring) [39]. Prawer et al. conducted a field study of how gamification can promote the creation of Javadoc software documentation in an agile environment [31]. LaToza et al. explored how gamification can play a role in crowdsourcing development work [24]. Duarte et al. explored how gamification can enhance requirements elicitation [14]. Gamification is also making its way into mainstream development environments such as Visual Studio [5].

To date, few studies have evaluated gamification while even fewer (as of 2014, only six) have investigated how gamification impacts developer motivation in industrial settings [29]. In particular, we see a lack of theoretical foundations to prescribe and evaluate how gamification can play a role in software development. However, this is not surprising as its application in this domain is still rather new.

In framing our research, we could not find any studies on how gamification impacts technical debt in practice, although Dubois et al.’s preliminary study showed that gamification for avoiding technical debt (such as code smells) seems to motivate students in an educational setting [15] and a white paper by Cognizant claims that the gamification of Sonar in a project reduced quality costs [7]. These preliminary studies provide some evidence that gamification can help address technical debt, but we lack empirical findings on how it may influence developer behaviour and motivation in an industrial setting. We next describe our design study where gamification was explored as a solution to manage technical debt.

III. RESEARCH METHODOLOGY

Applying gamification to reduce or avoid technical debt is a relatively new research direction and there are few insights on how gamification may impact the management of technical debt in software development. Given the lack of research in this area, our goal is to answer the following exploratory research questions:

- **RQ1**: How does gamification impact developer behaviour towards technical debt management?
- **RQ2**: How can managers use gamification to help them monitor and drive developers’ actions on technical debt?

Since our research objective was to design an artifact—a gamification tool—to motivate developers to reduce technical debt, we frame our study using the terminology and structure of a design study methodology [34]. In this section, we briefly present what a design study methodology is, introduce our industrial partner and discuss how we worked with them to characterize the problem we aimed to address.

A. Design Study Methodology

According to Selmaire et al. [34], a design study must first characterize the problem to be solved through a designed artifact (tool)—this step is done in collaboration with the identified users of the tool. The next step is to iteratively design and implement the artifact with ongoing input from the users. The tool design is validated using empirical methods and then the researchers reflect on the design study process as well as consider how the findings may be transferable to other settings. As feedback is gathered iteratively during the creation of the tool or artifact, it is expected that the problem characterization may need to be refined.

B. Study Partner Company

Our study involved a French governmental agency, named Pôle Emploi, that provides financial aid for unemployed people (5.5 million people in February 2017). Pôle Emploi has 50,000 employees among 900 offices in France and a website that receives over 45 million visits each month. The business depends on a software platform composed of several applications, maintained daily by 300 developers. Our study focuses on a central application of the platform, which we will refer to as C-App. C-App is highly strategic as it has a major financial impact on users. It was initially deployed in 2006 and now consists of 550k lines of code. It is a J2EE application (with Java 1.6) that is composed of two main components addressing different but related sets of functionalities. The C-App developers use the Eclipse IDE and other tools that are chosen by the company. The source code is hosted on a large, centralized version control system and each release has its own branch within the repository. Developers are allowed to commit directly to the repository and they conduct face-to-face code reviews with their peers before committing code.

---

[1] https://channel9.msdn.com/Achievements/VisualStudio
[2] https://www.cognizant.com/InsightsWhitepapers/Using-Gamification-to-Build-a-Passionate-and-Quality-Driven-Software-Development-Team.pdf
C-App is maintained by a group that includes 1 manager, 2 team leads, and 14 developers (divided into a team of 11 developers and a team of 3 developers). The group manager is responsible for the whole project—they define the main architecture and govern the group. Each of the team leads directly supervises one of the two C-App components and manages the corresponding development team. The project follows a scrum methodology and develops in three-week sprints. For each sprint, the group manager decides which evolution tasks to address and which incidents to fix.

C. Problem Characterization

In early December 2015, the C-App manager and team leads used the SonarQube [1] quality management platform to identify and measure TD. SonarQube revealed that the TD in C-App’s code would take an estimated 2,000 worker-days of development to address as there were over 15,000 code smells that required fixing. The manager then asked the team leads to encourage their developers to reduce TD as an underlying project objective. No other process steps were defined around TD at that time. SonarQube was somewhat successful in motivating developers to become aware of and address TD, but the high amount of debt (15K issues) that required fixing was demotivating. The developers also perceived fixing TD to be a boring and unrewarding task.

At the end of 2015, the company approached our research team because we had helped design a commercial technical debt management tool called Themis. At that time, Themis already linked commits with technical debt activities and supported some monitoring at a team level. However, we realized, as Pedreira et al. do, that: “Many software engineering tasks, such as testing and maintenance, are considered somewhat ‘destructive’ and not very appealing; i.e., this type of work is not intrinsically motivating, so specific mechanisms to foster motivation are needed.” [29] Therefore, we decided to explore the addition of gamification to Themis, anticipating that it may help reduce technical debt by changing developer behaviour and motivation. Themis (with the addition of gamification) was deployed at the company in April 2016 with a day of training for developers. A new version was deployed in mid-June to improve the scores computation and to provide some new features requested by the users, who were kept in the loop while developing the gamification of Themis. The following section describes the version of Themis currently used by our industry partner.

IV. THE THEMIS SOLUTION

To help our C-App industrial partner address their problems with technical debt management, we chose to customize and extend the commercially available Themis [3] technical debt management tool. This section discusses the customized version of the tool that we deployed after several iterations of design and feedback with the C-App development group.

As a technical debt management tool, Themis combines the information produced by a set of linters with the information contained in the version control system (VCS) to identify whose code is breaking or adhering to the linters’ rules. When a violation is detected, Themis warns the offending code’s author and points them to the problematic code. The gamification layer we added to Themis awards a score to developers based on the code they commit. It uses that score to create friendly competition and incentivize people to better manage their TD.

Below, we explain the core principles behind Themis and then we present the gamification layer and describe how managers and developers use it to work with TD.

A. Design Principles

Themis works together with version control systems and linters, expanding the features they provide by identifying which developers are adhering to or violating rules when they commit code. It performs an analysis of the commits recorded in the VCS to measure their impact on the rules and to link the commits to their corresponding authors. A commit can yield several positive or negative actions. A negative action is created for each rule violation that is triggered by a commit and a positive action is created for each rule violation that is removed by a commit. The actions extracted from a commit are assigned to the author of the commit.

As an example, Listing 2 shows a piece of code committed by Bob that violates the no-unreachable rule (line 4, see Section II-A). Themis sees that the code contains one negative action targeting the rule and assigns it to Bob.

Later, Alice made a commit (see Listing 3) that removes the aforementioned rule violation. Themis sees that this commit contains one positive action targeting the rule and assigns it to Alice.

This concept of positive or negative action is at the core of Themis’ gamification layer.

B. Gamification in Themis

The Themis gamification layer provides game elements and rules for “playing the game”, and has its own game design. Themis hosts a contest between developers where the goal is to be ranked as high as possible. Ranking is determined using a score computed from the actions performed by the
developers. Points are rewarded for each action and the sum of the points defines the score. Managers control the number of points scored for a given action, serving as "game masters". In our previous example configuration that assigns +1 to positive actions and −1 to negative actions, the score would be 1 for Alice and −1 for Bob. Therefore, Alice would have the best ranking and Bob would be in second position.

*Themis* uses a few key game elements. Each time a developer performs an action, *Themis* provides feedback so that the developer knows the impact of their actions on the score. A developer can also use this information to better understand the reasons for a score and possibly improve their coding activities. *Themis* also shows developer ranking on a leaderboard. Scores are visible by everyone and a person’s ranking is updated each time a new action is performed. Associating ranking with self-reputation encourages developers to pay better attention to TD and improve their coding.

The game design used by *Themis* consists of a timed contest. For our study, a contest lasted for the duration of a sprint (3 weeks, see Section III-B) and the score was reset each time a new sprint started. We note that the option of resetting the score was requested by our industrial partner. In the first *Themis* iteration we deployed, this feature was not supported. As a consequence, the contest never ended which was a mistake in the game design as it could be demotivating.

*Themis* also provides its users with challenges (c.f., Section IV-C). Managers are able to suggest developers perform or avoid certain actions in a specific time window. Depending on the outcome of the challenge, a bonus or penalty is awarded to all the developers in the group.

### C. Manager Views

Managers serve as "game masters" and do not participate in *Themis* contests. They configure the points associated with actions and define the challenges—action plans—given to developers. *Themis* provides a view where managers can observe all actions performed by the developers. This view also shows which rules are being violated the most and the score of each developer, useful information when configuring the points associated with actions and defining challenges.

*Themis* provides a view for configuring the points associated with actions. For each action, managers can configure whether the action is positive or negative and include information about the rule that is associated with the action (such as severity, category, or name). For example, a manager can configure the system so that positive actions award 2 points and negative actions deduct 2 points by default. They can further customize it so that negative actions that target the log.md rule deduct 5 points and positive ones award 10 points.

*Themis* provides a view for defining action plans to guide developers in managing specific TD. An action plan is time-boxed, can be assigned to one or more developers, and contains several objectives regarding the actions that should be performed or avoided. For instance, an action plan could include the following objectives: "perform less than five negative actions" and "perform ten positive actions regarding the log.md rule". An action plan specifies the bonus that will be awarded should the developer succeed or the penalty they will receive should they fail. When an action plan is assigned to a group, the bonus or penalty is given to all the developers in the group even if only one developer performed the work. The purpose of this is to encourage group collaboration. Figure 1 shows the view where managers can configure action plans.

### D. Developer Views

Developers participate in contests by performing actions. *Themis* provides a variety of views (see Figure 2) that developers can use to see their placing within a contest and review the actions they have performed:

- A *newsfeed* that presents the developer’s last actions with their associated points and rules. Anonymized versions of other developers’ actions are also shown.
- A *leaderboard* that presents the scores and rankings of all developers in the contest.
- A *dashboard* of all the actions performed by the developer with a full description of their associated points, rules, and the files that were affected.
- A list of ongoing action plans that have been assigned to the developer by the managers.

In addition to these views, *Themis* provides a suggestion module (shown in Figure 3) that suggests ways for the developer to score points by pinpointing actions that should be easy for them to perform. For example, this module identifies and visualizes (using a treemap) any rules that were violated in the files modified by the developer’s last commits, or which parts of the source code can provide the highest reward.
Developers can use this information to quickly and easily score points, thereby improving their ranking.

V. VALIDATION

To validate the final design of the gamified version of Themis, we conducted a survey with developers and managers to discern how the tool influenced their behaviours. We present our data collection and analysis methods, reflect on the limitations of our research process and present our findings.

A. Methods

We distributed two different survey questionnaires to members of the C-App team 3 months after Themis was first deployed: one version of the survey was sent to the manager and both team leads (referred to as managers in the rest of the paper) and another version was sent to the 14 developers. All 3 managers as well as 8 out of the 14 developers responded to the questionnaires. All answers were provided anonymously (unless participants chose to give us their email address) and participants were not given any incentive to answer the questionnaire—they generously spent time answering questions without compensation to help us understand the effects of the tool they were using.

The managers’ questionnaire included closed- and open-ended questions inquiring about the information they find useful (and why) from the different views available to them, how they use this information, and what feedback they received from their developers. The developers’ questionnaire also focused on the different views provided by Themis, asking how (and how often) they use the different features. Other developer questions were related to how important the score is to them, what steps they take to improve their score, and whether they noticed a change in their motivation to reduce TD since the introduction of Themis. Both questionnaires also asked whether participants think Themis had an impact on the TD of their project and what this impact was.

After manually translating the responses from French to English, we used coding to analyze the answers of our questionnaire which consists of labeling data to “quickly find, pull out, and cluster the segments relating to a particular research question, hypothesis, construct, or theme.” [27] We performed an initial coding cycle using provisional codes [27], i.e., a “start list” of codes matching the list of TDM activities described by Li et al. [26] as well as two codes related to extrinsic and intrinsic motivation, respectively.

To reduce bias during the validation process, we asked two independent researchers to review our survey questions. We also recruited an independent researcher (experienced in qualitative data analysis) to independently review our codes and coding. This independent review led to several iterations of the coding.

Through follow-up questions sent to willing participants, we were able to further verify that the insights we gained (i.e., the main themes) from our analysis of the survey responses resonated with the research participants. The answers to these questions helped to confirm our findings from the survey. They also provided additional insights we did not initially probe about in the two sets of questionnaires.

B. Research Limitations

There are inevitably a number of limitations with any study, some of which are specific to our chosen research methodology. Before presenting our findings, we discuss the limitations and the steps we took to offset them.

Throughout our design study, the tool researchers and designers were actively involved in the development and evaluation of the tool. However, we recognize that this active role of the researcher may have positively influenced the attitudes of the developers and managers towards the tool in the survey and follow-up questions. It even may have changed how they used the tool. This limitation is an artifact of our research methodology as the role “of the researcher is central and desirable [to a design study], rather than being a dismaying incursion of subjectivity that is a threat to validity.” [32] Indeed, the close knowledge of the teams and their needs informed the design of the tool so that it would solve their specific problem and it influenced the nature of the questions
C. Findings

Here, we report the findings from the survey and follow-up questions to provide insights on how Themis’ gamification features impacted developer behaviour and how managers made use of gamification for technical debt management.

Some answers from the closed-ended questions are first summarized in Figure 4. Starting with a list of provisional codes to analyze our survey answers, we established and iterated upon a codebook containing 34 codes. Table I lists the codes that are applicable to more than half of the participants; the full list of codes is available in our online supplementary materials. Using the codebook, we organized our findings according to seven themes as listed in Table II. We present below our findings, organized according to these themes which are shown in bold while codes are shown in SMALL CAPS. When quoting participants, their id appears as [m_i] or [d_i] for managers and developers, respectively.

Themis promotes technical debt reduction: When asked about the overall impact of Themis on TD REDUCTION, all developers and managers agreed that it had a positive impact. Even developers who expressed criticism and did not notice an increase in their own motivation recognized that gamification had a positive impact on other developers. One of the developers commented that “some seek to be on the top of the leaderboard.” [d_3] Another respondent however expressed a contrasting view by stating that Themis did not have a significant impact on TD reduction as “actions to reduce TD were already performed before; there haven’t been an increase in actions since.” [d_7] Our survey was performed three months after Themis was first deployed, which makes TD reduction “difficult to quantify for the time being.” [m_1] Although some data regarding the amount of TD was made available to us and a constant decrease of TD was observed, a statistical analysis of this data would be unreliable due to

---

1. http://promyze.com/themis
important confounding factors. The company had migrated their development infrastructure to new servers and operating systems and they implemented configuration changes in Sonar which impacted TD measurement.

**Themis made developers more attentive to the quality of the code they write:** Although Themis was initially designed to help reduce existing TD backlog and focus on TD repayment, our results show that it had a significant impact on TD PREVENTION leading to higher QUALITY STANDARDS. One developer mentioned paying “specific attention every day before committing code” [d2] as well using “quality measurement tools available to [them] before committing [their] code.” [d6]

The main EXTRINSIC factor that motivates developers is that their points will decrease on the leaderboard if they add TD: “There is less TD created because it is visible by others through the leaderboard, so we are more careful.” [d6] Moreover, when asked which part of their score they feel is most important, all developers indicated that they focus on having a low number of negative points: “You can have 0 points and be a developer applying the ‘clean code’ rules by the book, not generating any defects: that is the goal to reach because ultimately there must not be any TD.” [d5]

**INTRINSIC MOTIVATION** stemming from external motivators was expressed by developers as they focus on improving their own code: one developer explicitly told us that he “mostly look[s] to improve [himself] and having fewer negative points helps with that.” [d1] Another mentioned he uses the leaderboard for SELF-EVALUATION, to “position [himself] relative to [his] colleagues, not with the goal to show that [he is] better than them, but to see if [he is] as good as them.” [d6]

Extrinsic rewards are not effective for all participants in improving intrinsic motivation. As shown in Figure 2, two participants said that their score was not important to them and one of them strongly criticized gamification: “I don’t really like the idea of a leaderboard. [...] I think it is a pity that you need to have a reward (points) to code properly.” [d3] The fact that GAMIFICATION IS NOT NEEDED FOR ALL DEVELOPERS

---

**TABLE I**

| Code                          | Description                                                                 | Participants |
|-------------------------------|-----------------------------------------------------------------------------|--------------|
| TD EVOLUTION MONITORING      | *Themis* provides a way to monitor actions affecting the evolution of technical debt | 11           |
| POSITIVE IMPACT ON TD REDUCTION | *Themis* has a positive impact on TD reduction (accelerates TD reduction)     | 10           |
| TD PREVENTION*               | *Themis* helps prevent potential TD from being incurred                      | 10           |
| TD REPAYMENT*                | *Themis* helps developers resolve existing TD in a software system           | 9            |
| TD PRIORITIZATION*           | *Themis* ranks identified TD according to certain predefined rules           | 7            |
| QUALITY STANDARDS            | Participants are more attentive to quality standards and processes because of *Themis* | 7            |
| LIMITATION: “ONE SIZE FITS ALL” | The tool might not have the same impact on all participants or on other teams | 7            |
| EXTRINSIC MOTIVATION*        | Participants are motivated by rewards                                        | 7            |
| LEADERBOARD                  | Participants want to progress up the leaderboard                             | 7            |
| TD MEASUREMENT*              | *Themis* helps quantify and estimate the level of overall TD                | 6            |
| INTRINSIC MOTIVATION*        | Participants are motivated by their own self-improvement                     | 6            |

**TABLE II**

| Theme                                | Answers fitting the theme | Discrepant answers |
|--------------------------------------|---------------------------|--------------------|
| *Themis* promotes technical debt reduction | m1, m2, m3, d1, d2, d3, d4, d5, d6, d7, d8 | m2, d7             |
| *Themis* made developers more attentive to the quality of the code they write | m2, m3, d1, d2, d4, d5, d6, d8 | d3, d7             |
| Developers appreciate and follow the suggestions provided by *Themis* | d1, d3, d5, d6, d8 | d7                 |
| Developers follow an opportunistic approach to TD repayment | d4, d6, d8 |             |
| *Themis* provides monitoring and awareness of TD for individuals and teams | m1, m2, m3, d1, d2, d3, d4, d5, d6, d7, d8 |             |
| *Themis* allows managers to adapt TDM to the context of their projects | m1, m2, m3 | d4, d6             |
| *Themis* promotes ongoing discussion about technical debt between stakeholders | m1, m2, m3, d6 |             |
Developers appreciate and follow the suggestions provided by Themis: Although TD prevention remains the most important in the eyes of participants (see Figure 4), TD REPAYMENT was mentioned by all developers and managers as a TDM activity where Themis is helpful. Most developers also follow the TD PRIORITIZATION of suggestions shown in Themis, with five out of eight developers using it at least once a week. One developer mentioned that he looks at the goals defined by managers “to see how make quick progress on the leaderboards.” [d1] Developers who are not directly motivated by gamification also “[use Themis] to look for files containing several anomalies to be fixed.” [d3]

One developer in contrast indicated that he does not take part in large TD repayment activities and that “making mass TD repayment actions is absolutely not a part of [his] work. It is uninteresting and there is some development that is way more important that needs to be done.” [d7] He also pointed to a possible side effect of gamification: “some people spend time fixing classes that are historically not modified only to win a few points when they have pending development/debugging.” [d7] This point was acknowledged by managers who assured us that this view is incidental but they remain careful about it.

Developers follow an opportunistic approach to TD repayment: One strategy we observed to reduce TD is what we call OPPORTUNISTIC TD REPAYMENT: “[To improve my score in Themis.] I fix the content of a file that I have to modify for my development. NB: It is not Themis which recommended this file to me (but my development needs), and it is not Themis which pointed me to the existing defects in this file (but the quality measurement tools in my IDE).” [d6] For this particular strategy, the initial motivator is clearly the points reward, as this strategy was either mentioned when we asked participants about the steps they take to improve their score, or the participants themselves mentioned the reward as a goal.

Themis provides monitoring and awareness of TD for individuals and the team: Developers and managers indicated that they use Themis to monitor positive and negative actions performed. Developers mainly use the monitoring features of the main dashboard to “evaluate [their] work very quickly” [d4] and “see what are [their] areas of improvement” [d4]. Managers also strive for “aggregate information allowing [them] to do medium/long term monitoring.” [m1] They also use MONITORING to update TD PRIORITIZATION and to COMMUNICATE with their developers: “the action reports at the end of the sprint […] are useful to me in order to create action plans, or to provide reminders to the team or individuals, if needed.” [m3]

Themis allows managers to adapt TDM to the context of their project: Managers emphasized that “anomalies created and fixed by developers are strongly connected to the project’s context (age, architectural choices, . . . ).” [m2], and some rules may not be appropriate to the project or specific modules. An example shared in the follow up questions was the rule stating that “cycles between packages should be removed”, which is tied to initial architectural choices and would now be too costly to remove. This rule was subsequently disabled in Themis by the managers. Although this INTENTIONAL TD PERSISTENCE is supported by Themis, our follow up questions revealed that there are cases where intentional TD could not be ignored by Themis. These cases occur when editing code to comply to a rule—which is not disabled for the whole project—would in turn break another rule, which would require editing one or more different classes. Furthermore, the project uses non-regression testing further increasing the possible costs of modifying a new class. In this case, developers would rely on the fact that the score is periodically reset to undo the negative points caused by intentional TD. Alternatively, one respondent suggested that a manager could undo the negative points someone receives due to intentional debt.

Themis promotes ongoing discussion between stakeholders about technical debt: While using Themis, TD is actively discussed by managers and developers, thus increasing their awareness of TD. During development phases, “as soon as someone sees [newly created TD], they share it loudly in the office to ask the (anonymous) person responsible for it to repay it.” [d6] TD was also discussed extensively during configuration of the tool. Although configuring the amount of points attributed to each rule can only be done by managers, THE WHOLE TEAM WAS INVOLVED IN THE CONFIGURATION of Themis. Thanks to our follow up feedback from one developer, we better understand the process followed by the team to decide on the amount of points to assign to specific rules, and what kind of debates were initiated by Themis: “Different proposals were made (penalizing more TD creation but leaving repayment points low, penalizing more the creation of more severe defects and reward more the correction of severe defects, etc) and a vote was taken to decide which rule to set up. There are multiple arguments here: some consider that creating TD is worse and has to impact more the developer than the correction of an equivalent TD, others will allow the ‘right to make a mistake’ and consider that a developer who fixed his mistake must be in a neutral state” [d6]. The latter was adopted by the team. This example show that Themis puts TD at the center of stakeholders COMMUNICATION.

VI. Proposed Guidelines

Our research findings lead us to the premise that gamification can play a helpful role in technical debt management, both in reduction and in prevention. But applying it may not always be wise. Here we consider the transferability of our work (the primary aim of our study as mentioned earlier) by proposing a set of guidelines that practitioners and researchers may refer to should they wish to apply gamification for managing technical debt. Although we emphasize gamification for technical debt, we note that many of these principles may apply to other...
software engineering tasks (such as code review). Where relevant, we relate the guidelines to the literature, but we remind the reader that there is a dearth of theories or advice on how to use gamification for serious work in general [33] and even fewer theories for applying gamification in software engineering [29].

Guideline 1: Nurture a positive team culture

The managers and developers from our study were instrumental in setting a positive culture for the use of gamification. In particular, the team did not take “the game” too seriously and they enjoyed an atmosphere of playfulness, as one of the developers shared with us how: “[The leaderboard] allows to figure out who will bring chocolate croissants at the end of the sprint.” [d3] Rather than saying a certain developer was “last” they could joke that they owed the team chocolate croissants. In particular, we observed that the managers were sensitive to the possible drawbacks of gamification and they were careful not to misuse the information. In turn, the developers trusted their managers and each other. A trusting culture may not always be possible with a different management style.

Guideline 2: Tailor gamification to suit different developers

Just as different team cultures will influence how gamification impacts behaviour and motivation, the way gamification is used may need to be further adapted to suit different developer characteristics. As we saw from some of our survey responses, experienced developers may not feel the need for the extrinsic motivators, because they do a good job anyway and gamification could force them to use yet another tool on top of the many tools they already use [41].

Developer age and gender may be factors to consider, in our study the developers were all male and between the age of 25-34 and many on the team already played games. The success of gamification may have been in part due to age, as Dorling et al. note that Generation Y users appreciate clear goals, trackable progress and social rewards [13]. Furthermore, younger developers may be more influenced by money, whereas seasoned developers may be more motivated by task variety or challenge [19]. In terms of gender, Gneezy et al. [18] and Vasilescu et al. [42] found some differences in how females participate on the gamified StackOverflow environment. We may see differences in how females respond to gamification of technical debt. Personality is another consideration as one manager noted: “Some developers have a more discreet nature and gamification may not be a good motivator for these ones.” [m22]

Guideline 3: Adapt the game to project context

Varied project characteristics may impact which features are needed or should be avoided. For example, with new projects where new features are being added at a fast velocity, it may be more important to prevent technical debt: “It can be less interesting on a new project, since a clean code will not generate any points.” [d8] While for legacy projects, repayment may need more motivation: “The score is important for older projects with an important debt in order to create a competition and push developers to fix that debt.” [d8]

Generation of intentional debt, which we mentioned in our findings, is another example. There still are cases where developers are punished for TD they have to add to be consistent with existing TD that is too expensive to be repaid. When such situations arise, it may be beneficial for managers and developers to manually adjust scores to nurture a feeling of fairness and to maintain morale. Currently, this pitfall is mitigated by score resets at the end of every sprint. This customization was probably essential for the success of Themis as previously developers felt overwhelmed as once behind they could not catch up.

Guideline 4: Aim for seamless integration with existing processes and tools

Careful integration of a new tool within a developer’s workflow is critical [20]. The gamification aspects here are smoothly integrated with the developers’ workflow and existing version control and analysis tools. Furthermore, Themis was designed to be easy to use and learn. These design aspects are critical as Gartner et al. warn about falling prey to this pitfall: “80% of the gamified applications will fail to meet their business goals due to a poor design” [12].

Guideline 5: Keep users in the loop

Users of a gamified tool should be involved, if possible, in both designing the game and customizing its rules. Keeping everyone involved during the design of the tool is considered an examplar strategy to prevent negative side-effects of gamification, and to make users committed to the design of the game [2].

Guideline 6: Monitor how extrinsic motivators influence intrinsic motivation over time

Tailoring to developer characteristics is important but it is also important to monitor changes in motivation over time, as the game may need further configuration. How gamification may impact other management goals over time should also be monitored: “managers must provide challenging problem-solving tasks, explicitly recognize quality work, and give developers autonomy to do their jobs. Managing these factors effectively will engage developers and excite them in their work.” [19]

Guideline 7: Consider when not to gamify

Although gamification shows potential benefits for developers and managers, in agreement with other researchers, we do not advocate that it should be blindly applied as “lemming engineering” [29]. There may be other risks to consider—if developers are busy playing the “game”, what activities does it replace? Will it lead to a lot of code TD repayment, but ignore
architectural TD? These kinds of strategic issues should be carefully considered.

VII. FUTURE WORK AND CONCLUSIONS

The use of gamification in software engineering is becoming quite prevalent, in part due to an increased emphasis on data science in software engineering as well as an increase in the use of social media. But the introduction of some seemingly rather innocent features can have a strong impact. We feel that much more research is needed into the benefits and risks of gamification, while at the same time there are more tasks in software engineering where gamification could be introduced.

We investigated how gamification could support technical debt management activities, but our study is just the first step in this research. The main outputs from our study are a novel tool design and a set of guidelines and we anticipate that these guidelines can be extended and then used as preliminary propositions in building a theory of the role of gamification for managing technical debt in future research.

In closing, we make a call for more research studies on gamification. We plan to conduct a longitudinal study of gamification use, which could be very insightful. Will gamification succeed in reducing TD and supporting TDM activities over one or two years of a project? We further agree with Pedreira et al. that there is a need for comparative studies and that we should strive to conduct studies of developers doing the same task in a gamified and a non-gamified manner. However, we note that doing so is very difficult due to many possible confounds—some of which we alluded to in the guidelines. In the meantime, we hope that our findings from this study will prove useful to both researchers and practitioners interested in the role of gamification in software engineering.

Acknowledgements: We thank our research participants and Cassandra Petrachenko for improving our paper.

VIII. REFERENCES

REFERENCES

[1] SonarQube™
[2] A. Algashami, A. Shahri, J. McAlaney, J. Taylor, K. Phalp, and R. Ali. Strategies and Design Principles to Minimize Negative Side-Effects of Digital Motivation on Teamwork. In Persuasive Technology: Development and Implementation of Personalized Technologies to Change Attitudes and Behaviors, pages 267–278. Springer, Cham, Apr. 2017.
[3] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spinola. Towards an Ontology of Terms on Technical Debt. In 2014 Sixth International Workshop on Managing Technical Debt (MTD), pages 1–7, Sept. 2014.
[4] K. Beck, M. Fowler, and G. Beck. Bad smells in code. Refactoring: Improving the design of existing code, pages 75–88, 1999.
[5] B. Biegel, F. Beck, B. Lesch, and S. Diehl. Code Tagging as a Social Game. In 30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014, pages 411–415, 2014.
[6] B. Brathwaite and I. Schreiber. Challenges for Game Designers. Charles River Media, Inc., Rockland, MA, USA, 1 edition, 2008.
[7] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazzourka. Managing Technical Debt in Software-reliant Systems. In Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, FoSER ’10, pages 47–52, New York, NY, USA, 2010. ACM.
[8] W. Cunningham. The WyCash Portfolio Management System. In Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum), OOPSLA ’92, pages 29–30, New York, NY, USA, 1992. ACM.
[9] B. Curtis, J. Sappidi, and A. Szynkarski. Estimating the Principal of an Application’s Technical Debt. IEEE Software, 29(6):34–42, 2012.
[10] T. Dal Sasso, A. Mocci, M. Lanza, and E. Mastroddiaca. How to gamify software engineering. In Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on, pages 261–271. IEEE, 2017.
[11] E. L. Deci, R. Koestner, and R. M. Ryan. A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation. Psychological Bulletin, 125(6):627–668, 1999.
[12] 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: Envisioning Future Media Environments, MindTrek ’11, pages 9–15, New York, NY, USA, 2011. ACM.
[13] A. Dorling and F. McCaffery. The gamification of SPICE. In International Conference on Software Process Improvement and Capability Determination, pages 295–301. Springer Berlin Heidelberg, 2012.
[14] D. Duarte, C. Farinha, M. M. da Silva, and A. R. da Silva. Collaborative requirements elicitation with visualization techniques. In Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on, pages 343–348. IEEE, 2012.
[15] D. J. Dubois and G. Tamburrelli. Understanding Gamification Mechanisms for Software Development. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pages 659–662, New York, NY, USA, 2013. ACM.
[16] D. Falesi, P. Kruchten, and P. Avgeriou. Introduction to the special issue on technical debt in software systems. Journal of Systems and Software, 2016.
[17] M. Fowler. TechnicalDebtQuadrant, 2009.
[18] U. Gneezy, M. Niederle, A. Rustichini, and others. Performance in competitive environments: Gender differences. QUARTERLY JOURNAL OF ECONOMICS-CAMBRIDGE MASSACHUSETTS, 118(3):1049–1074, 2003.
[19] T. Hall, H. Sharp, S. Beecham, N. Baddoo, and H. Robinson. What do we know about developer motivation? IEEE software, 25(4):92, 2008.
[20] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why Don’t Software Developers Use Static Analysis Tools to Find Bugs? In Proceedings of the 2013 International Conference on Software Engineering, ICSE ’13, pages 672–681, Piscataway, NJ, USA, 2013. IEEE Press.
[21] J. Juul. The game, the player, the world: looking for a heart of gameness. In Digital Games Research Conference 2003, 4-6 November 2003, University of Utrecht, The Netherlands, 2003.
[22] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. The emerging role of data scientists on software development teams. In Proceedings of the 38th International Conference on Software Engineering, pages 96–107, 2016.
[23] P. Kruchten, R. L. Nord, and I. Ozkaya. Technical Debt: From Metaphor to Theory and Practice. IEEE Software, 29(6):18–21, 2012.
[24] T. D. LaToza, W. B. Towne, A. Van Der Hoek, and J. D. Herbelsb. Crowd development. In Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on, pages 85–88, Citeseer, 2013.
[25] J. L. Letouzey and M. Ilkiewicz. Managing Technical Debt with the SQALE Method. IEEE Software, 29(6):44–51, Nov. 2012.
[26] Z. Li, P. Avgeriou, and P. Liang. A systematic mapping study on technical debt and its management. Journal of Systems and Software, 101:193 – 220, 2015.
[27] M. B. Miles, A. M. Huberman, and J. Saldana. Qualitative data analysis: A methods sourcebook. SAGE Publications, Incorporated, 2013.
[28] E. B. Passos, D. B. Medeiros, P. A. S. Neto, and E. W. G. Clua. Turning Real-World Software Development into a Game. In Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on, pages 260–269, Nov. 2011.
[29] O. Pedroira, F. Garcia, N. R. Brsaboa, and M. Piattini. Gamification in software engineering - A systematic mapping. Information & Software Technology, 57:157–168, 2015.
[30] http://www.hanselman.com/blog/GitHubActivityGuiltAndTheCodersFitBit.aspx
[30] T. M. Pigoski. Practical Software Maintenance: Best Practices for Managing Your Software Investment. Wiley Publishing, 1st edition, 1996.

[31] C. R. Prause, J. Nonnen, and M. Vinkovits. A field experiment on gamification of code quality in agile development. In Psychology of Programming Interest Group Annual Conference (PPIG), volume 2012, 2012.

[32] B. Reeves and J. L. Read. Total Engagement: Using Games and Virtual Worlds to Change the Way People Work and Businesses Compete. Harvard Business School Press, 1 edition, Nov. 2009. Published: Hardcover.

[33] K. Seaborn and D. I. Fels. Gamification in theory and action: A survey. International Journal of Human-Computer Studies, 74:14–31, Feb. 2015.

[34] M. Sedlmair, M. Meyer, and T. Munzner. Design study methodology: Reflections from the trenches and the stacks. IEEE Transactions on Visualization and Computer Graphics, 18(12):2431–2440, 2012.

[35] S. Sheth, J. Bell, and G. Kaiser. A Gameful Approach to Teaching Software Design and Software Testing. Computer Games and Software Engineering, 9:91, 2015.

[36] S. Sheth, textbf{Bell}, J, and G. Kaiser. HALO (Highly Addictive, socially Optimized) Software Engineering. In Proceeding of the 1st international workshop on Games and software engineering, GAS ’11, 2011.

[37] C. A. Siebra, R. G. Oliveira, C. B. Seaman, F. Q. B. Silva, and A. L. M. Santos. Theoretical conceptualization of TD: A practical perspective.

[38] L. Singer and K. Schneider. It Was a Bit of a Race: Gamification of Version Control. In Proceedings of the Second International Workshop on Games and Software Engineering: Realizing User Engagement with Game Engineering Techniques, GAS ’12, pages 5–8, Piscataway, NJ, USA, 2012. IEEE Press.

[39] W. Snipes, V. Augustine, A. R. Nair, and E. Murphy-Hill. Towards Recognizing and Rewarding Efficient Developer Work Patterns. In Proceedings of the 2013 International Conference on Software Engineering, ICSE ’13, pages 1277–1280, Piscataway, NJ, USA, 2013. IEEE Press.

[40] F. Steffens, S. Marczak, F. Figueira Filho, C. Treude, L. Singer, D. Redmiles, and B. Al-Ani. Using Gamification as a Collaboration Motivator for Software Development Teams: A Preliminary Framework. In Brazilian Symposium on Collaborative Systems (SBSC), 2015.

[41] M. A. Storey, A. Zagalsky, F. Filho, L. Singer, and D. German. How social and communication channels shape and challenge a participatory culture in software development. IEEE Transactions on Software Engineering, PP(99):1–1, 2016.

[42] B. Vasilescu. Human aspects, gamification, and social media in collaborative software engineering. In Companion Proceedings of the 36th International Conference on Software Engineering, pages 646–649. ACM, 2014.

[43] J. Yli-Huumo, A. Maglyas, and K. Smolander. How do software development teams manage technical debt? – An empirical study. Journal of Systems and Software, pages 1 – 24, 2016.