HOYAMA MARIA DOS SANTOS

CLEANGAME:

UMA ABORDAGEM GAMIFICADA PARA DETECÇÃO DE CODE SMELLS

LAVRAS – MG

2019

HOYAMA MARIA DOS SANTOS

CLEANGAME:
UMA ABORDAGEM GAMIFICADA PARA DETECÇÃO DE CODE SMELLS

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Engenharia de Software, para obtenção do título de Mestre em Ciência da Computação.

Prof. Dr. Rafael Serapilha Durelli (DCC/UFLA)
Orientador

Prof. Dr. Maurício Ronny de Almeida Souza (DCC/UFLA)
Coorientador

LAVRAS – MG

2019

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Santos, Hoyama Maria dos.
CleanGame : Uma abordagem gamificada para detecção de code smells / Hoyama Maria dos Santos. - 2019.
86 p. : il.

Orientador(a): Rafael Serapilha Durelli.
Coorientador(a): Maurício Ronny de Almeida Souza.
Dissertação (mestrado acadêmico) - Universidade Federal de Lavras, 2019.
Bibliografia.

1. Educação em Engenharia de Software. 2. Gamificação. 3. Code Smells. I. Durelli, Rafael Serapilha. II. Souza, Maurício Ronny de Almeida. III. Título.

HOYAMA MARIA DOS SANTOS

CLEANGAME: UMA ABORDAGEM GAMIFICADA PARA DETECÇÃO DE CODE SMELLS

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Engenharia de Software, para obtenção do título de Mestre em Ciência da Computação.

APROVADA em 12 de dezembro de 2019.

Prof. Dr. Rafael Serapilha Durelli	UFLA
Prof. Dr. Paulo Afonso Parreira Junior	UFLA
Prof. Dr. Maurício de Almeida Souza	UFLA
Prof. Dr. Eduardo Magno Lages Figueiredo	UFMG

Prof. Dr. Rafael Serapilha Durelli (DCC/UFLA)
Orientador

Prof. Dr. Maurício Ronny de Almeida Souza (DCC/UFLA)
Co-Orientador

LAVRAS – MG
2019

RESUMO

Refatoração é o processo de transformar a estrutura interna do código existente sem alterar seu comportamento observável. Muitos estudos mostraram que a refatoração aumenta a facilidade na manutenção e a compreensibilidade do programa. Devido a esses benefícios, é reconhecida como uma prática recomendada na comunidade de desenvolvimento de software. No entanto, antes das atividades de refatoração, os desenvolvedores precisam procurar essas oportunidades, ou seja, serem capazes de identificar *code smells*, que são essencialmente instâncias de *design* inadequado e opções de implementação mal consideradas que podem prejudicar a capacidade de manutenção e a compreensão do código. No entanto, a identificação do *code smells* ainda é negligenciada no currículo de Ciência da Computação, recentemente, os educadores de Engenharia de Software iniciaram a exploração da gamificação, o que implica o uso de elementos de jogo em contextos não relacionados, para melhorar os resultados instrucionais em contextos educacionais. O potencial da gamificação reside em apoiar e motivar os alunos, aprimorando o processo de aprendizagem e seus resultados. Desta forma, este trabalho irá avaliar até que ponto esta alegação é válida no contexto do reforço pós-treinamento. Para esse fim, foi implementado e desenvolvido o CleanGame, que é uma ferramenta gamificada que cobre um aspecto importante do currículo da refatoração: identificação do *code smells*, para fins de validação foi realizado um experimento envolvendo 18 participantes para investigar a eficácia da gamificação no contexto do reforço pós-treinamento. Assim obteve-se como resultado, que os participantes, em média, conseguiram identificar o dobro de *code smell* durante o reforço da aprendizagem com uma abordagem gamificada em comparação com uma não gamificada. Além disso, foi administrada uma pesquisa de atitude pós-experimento aos participantes, em que a maioria demostrou uma atitude positiva com relação ao CleanGame.

Palavras-chave: Refatoração, gamificação, *code smell*, educação em engenharia de software, reforço pós-treinamento

ABSTRACT

Refactoring is the process of transforming the internal structure of existing code without changing its observable behavior. Many studies have shown that refactoring increases program maintainability and understandability. Due to these benefits, refactoring is recognized as a best practice in the software development community. However, prior to refactoring activities, developers need to look for refactoring opportunities, i.e., developers need to be able to identify code smells, which essentially are instances of poor design and ill-considered implementation choices that may hinder code maintainability and understandability. However, code smell identification is overlooked in the Computer Science curriculum. Recently, Software Engineering educators have started exploring gamification, which entails using game elements in non-game contexts, to improve instructional outcomes in educational settings. The potential of gamification lies in supporting and motivating students, enhancing the learning process and its outcomes. We set out to evaluate the extent to which such claim is valid in the context of post-training reinforcement. To this end, we devised and implemented CleanGame, which is a gamified tool that covers one important aspect of the refactoring curriculum: code smell identification. We also carried out an experiment involving eighteen participants to probe into the effectiveness of gamification in the context of post-training reinforcement. We found that, on average, participants managed to identify twice as much code smells during learning reinforcement with a gamified approach in comparison to a non-gamified approach. Moreover, we administered a post-experiment attitudinal survey to the participants. According to the results of such survey, most participants showed a positive attitude towards CleanGame.

Keywords: Refactoring, gamification, code smell, Software Engineering education, post-training reinforcement

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A partir do momento que um sistema de software é finalizado, sua fase de manutenção inicia-se (MALHOTRA; CHUG, 2016). Esta etapa é essencial para manter e adequar um software de acordo com as necessidades do usuário (ALVES; MARTINS; PAULISTA, 2017), e seu custo é definido de acordo com a capacidade de manutenção do mesmo (MALHOTRA; CHUG, 2016).

Neste contexto, um código bem projetado é fundamental para a manutenibilidade de sistemas. Assim, a refatoração é uma técnica importante para buscar a melhoria da qualidade interna do código, favorecendo a manutenção de um bom *design* (ALVES; MARTINS; PAULISTA, 2017). Dessa forma, é importante que desenvolvedores saibam identificar possíveis oportunidades de refatoração (AMORIM et al., 2014).

O termo "refatoração" foi introduzido por Opdyke em 1990 (OPDYKE, 1990) como um processo de reestruturação do código de software (MALHOTRA; CHUG, 2016). É, portanto, uma atividade de reorganização que visa melhorar a estrutura interna de um corpo de código existente, mantendo seu comportamento externo. Esta atividade objetiva a característica de qualidade de um sistema de software, incluindo extensibilidade, reutilização e eficiência (AMMAR; BHIRI, 2015).

Uma das questões-chave ao se realizar a refatoração é saber identificar possíveis problemas de *design*, ou seja, conseguir reconhecer os chamados *bad smells* ou *code smells* (LIU et al., 2015). Os *code smells*, ou apenas *smells*, são códigos que representam sintomas de *design* pobre e más escolhas de implementação, que dificultam a compreensibilidade e, possivelmente, aumentam a propensão a alterações e falhas (PALOMBA et al., 2014).

Dessa forma, aprender como identificar *code smells* é uma atividade importante para estudantes e/ou profissionais de TI [1]. Além disso, essa atividade de identificação não é trivial e demanda conhecimento (ITO et al., 2014).

De acordo com pesquisas recentes (SHARMA; SPINELLIS, 2018), diversos estudos, técnicas e abordagens foram/são utilizadas na literatura para auxiliar na identificação de *code smells*, por erxemplo: (i) abordagens baseada em métricas (CHIDAMBER; KEMERER, 1994; VIDAL; MARCOS; DÍAZ-PACE, 2016; MARINESCU, 2004; MARINESCU, 2005); (ii) abordagens que utilizam regras e heurísticas (SURYANARAYANA; SAMARTHYAM; SHARMA, 2014; MOHA et al., 2010); (iii) abordagens que utilizam histórico de um determinado software (PA-

[1] Tecnologia da informação.

LOMBA et al., 2015a); (iv) abordagens baseada em otimizações (MAIGA et al., 2012; KHOMH et al., 2009); e recentemente (v) abordagens que utilizam algoritmos de *machine learning* para criar modelos otimizados que auxiliam na identificação de *code smells* (OUNI et al., 2015).

Embora diversas pesquisas tenham sido conduzidas, Arcoverde, Garcia e Figueiredo (2011) conduziram uma pesquisa para entender como desenvolvedores reagem com a presunção de *code smells*. O resultado mostrou que desenvolvedores adiam a remoção dos *code smells* por dois principais motivos: *(1)* receio de executar a tarefa de forma equivocadamente e *(2)* receio de prejudicar a interface de execução do projeto.

Além disso, Peters e Zaidman (2012) também analisaram o comportamento de desenvolvedores no que diz respeito ao ciclo de vida de *code smells*. Resultados mostram que até mesmo quando os desenvolvedores estão cientes da presença de *code smells*, não executam qualquer tipo de refatoração. Sendo essas práticas frequentemente negligenciadas em favor de atividades mais aparentes, como *design* e implementação, passar pelo código enquanto se procura por *smells* é uma tarefa dispendiosa.

Um desafio recorrente na educação em Engenharia de Software é envolver os alunos em atividades de aprendizagem relacionadas às práticas profissionais. Muitas vezes é um obstáculo para os estudantes dessa área contextualizarem como alguns conceitos e habilidades podem se encaixar ou influenciar suas realizações futuras.

Recentemente, na esperança de lidar com esse desafio, a comunidade educacional de Engenharia de Software recorreu a estratégias pedagógicas inovadoras, como a gamificação (SOUZA et al., 2018; ALHAMMAD; MORENO, 2018). Essencialmente, a gamificação implica empregar elementos de *design* de jogos em um ambiente que não são jogos. Em outras palavras, a gamificação é centrada em gerar experiências de aprendizado que transmitem sentimentos e envolvem alunos como se estivessem jogando, mas não objetivando o entretenimento. Dessa forma, conjectura-se que a gamificação pode ser usada para melhorar a educação em Engenharia de Software. Mais especificamente, acredita-se que a gamificação pode ser usada para apoiar e motivar os alunos no desenvolvimento de habilidades de identificação de *code smells*, transformando uma atividade difícil e um tanto tediosa em uma experiência envolvente.

Com o objetivo de sanar essa limitação e apresentar o conteúdo de *code smells* de maneira interativa e pedagógica para os alunos e/ou profissionais, acredita-se que a gamificação pode ser uma tecnologia de extrema valia. Segundo Stegeman, Barendsen e Smetsers (2014),

uma melhor aprendizagem está ligada a mais *feedbacks*, o que é possível de maneira prática através de sistemas interativos de ensino.

Para um melhor entendimento do propósito deste projeto este capítulo traz o problema e motivação (Seção 1.1), os objetivos (Seção 1.2), a metodologia utilizada para o desenvolvimento do mesmo (Seção 1.3) e as contribuições e relevância do trabalho (Seção 1.4). Finalmente, na Seção 1.6 é descrita a organização deste documento.

1.1 Problema e Motivação

Atualmente, o processo de aprendizagem e aperfeiçoamento de técnicas tem caminhado para se tornar mais atraente às pessoas, uma vez que as novas gerações fazem uso intensivo de tecnologias digitais, esse envolvimento faz com que desprender um tempo jogando não seja uma tarefa tediosa, mas sim, uma atividade rotineira (FREITAS et al., 2017).

Pensando nisso, o processo de aprendizagem deve envolver as pessoas a experimentar as práticas profissionais de tal forma que entendam quais práticas e técnicas são úteis em diversas situações. Um profissional qualificado deve ter seu currículo planejado a fim de treiná-lo a prestar seus serviços, ou seja, o ensino deve ser ofertado de forma a garantir uma prática profissional bem sucedida. Para alcançar este objetivo, mudanças substanciais são necessárias, incluindo mudanças no modelo curricular, no sentido de superar o chamado currículo rígido para dar lugar a currículos semi-flexíveis, flexíveis ou modulares (IEEE,ACM, 2016).

Embora o uso de jogos na educação em Engenharia de Software não seja novo, e as tecnologias hoje disponíveis possibilitem novas oportunidades para usar jogos e seus elementos para melhorar a aprendizagem e envolvimento dos alunos (SOUZA et al., 2017), esse mecanismo de ensino é menos explorado na área de Engenharia de Software do que nas demais áreas como auxilio educacional (FIGUEIREDO et al., 2007).

Desta forma, a gamificação é inserida como uma ferramenta para aumentar o engajamento dos usuários, que interagem com o sistema como jogadores, competindo para obter a maior pontuação ao realizar o maior número de atividades no sistema. Já que a gamificação trata-se da aplicação de elementos de jogo em contextos diversos, ou seja, apenas os elementos são necessários, e não toda a organização do mesmo.

A escolha da utilização da gamificação para a aprendizagem e aprimoramento da identificação de *code smells* adveio da intenção de motivar desenvolvedores a aprenderem "jogando".

Já que, as pessoas são mais motivadas quando são capazes de interagir ativamente com determinado conteúdo, e não somente aprendendo de maneira tradicional (aulas expositivas).

Nesse caso, espera-se contribuir com uma melhor formação de profissionais com experiências práticas em identificação de *code smells*, para que sejam desenvolvedores melhores. Almeja-se que além da capacidade de refatorar, o desenvolvedor, no processo de desenvolvimento, já tenha consciência dos *code smells* que ele pode estar inserindo, e que um código com a presença de *smells* não lhe pareça apropriado.

Para aprendizes, a prática reside na identificação de *code smells* em trechos de códigos a partir de um oráculo oriundo de resultados de uma ferramenta de identificação de *smells*, ou de perguntas sobre *code smells* previamente cadastradas no sistema. Desta forma, o jogador deve responder a uma série de desafios propostos pelo sistema, utilizando o menor número de dicas possível, a fim de obter a maior pontuação. Assim, cria-se um ambiente orientado a prática para o desenvolvimento de habilidades relacionadas a inspeção de código, em relação às boas práticas de *design*.

Para ambientes profissionais, propõe-se que o sistema possa colaborar para a criação da cultura de inspecionar *design* do código periodicamente, com intuito de manter uma lista ativa de possíveis pontos de correção do sistema para eventuais refatorações, ordenada por prioridades definidas pelos próprios colaboradores. Contudo, é importante enfatizar que o objetivo principal deste projeto é o foco no aspecto educacional da ferramenta.

1.2 Objetivos

O objetivo deste trabalho é analisar a relevância do uso da gamificação no ensino, e aprimoramento da identificação de *code smells* em códigos. Um sistema gamificado foi implementado e através de um experimento foi avaliado o quanto o mesmo é pertinente para esse aprendizado. Para o alcance do objetivo geral, este trabalho considera os seguintes objetivos específicos (OE):

- **OE_1:** Investigar o uso de gamificação em ferramentas e sistemas para apoio ao ensino de Engenharia de Software;
- **OE_2:** Avaliar ferramentas de identificação de *code smells*;
- **OE_3:** Implementar o sistema utilizando os conceitos fundamentais da gamificação;

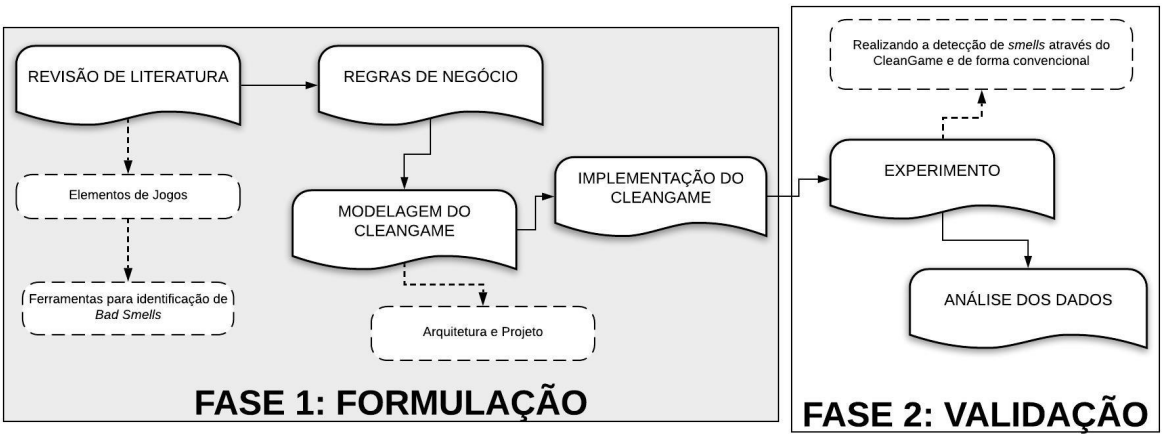
- **OE$_4$:** Avaliar o uso do sistema desenvolvido através de um experimento.

Além do objetivo central dessa pesquisa, pretende-se apresentar a metodologia de desenvolvimento do sistema gamificado implementado, colaborando assim, para o desenvolvimento de mais sistemas em diversificados seguimentos da Engenharia de Software.

1.3 *Design* da Estrutura de Trabalho

Esse seção descreve o desenho da pesquisa e os métodos planejados para seu desenvolvimento. A pesquisa foi dividida em duas fases e seis atividades, sendo a 1ª Fase (Formulação) composta por quatro atividades e a 2ª Fase (Validação) é composta por duas atividades.

Figura 1.1 – Etapas da Pesquisa



Fonte: Do autor (2019).

A Figura 1.1 apresenta o *design* da estrutura de trabalho dessa pesquisa. Ela foi organizada em quatro atividades na etapa de formulação: "Revisão da Literatura"(atividade 1); "Regras de negócio"(atividade 2); "Modelagem do sistema"(atividade 3); "Implementação CleanGame"(atividade 4); e duas atividades na etapa de validação: "Experimento"(atividade 5); "Análise dos dados"(atividade 6).

A primeira atividade consiste na revisão da literatura e objetivou mensurar qual o estado da arte, envolvendo gamificação no campo de educação em geral e em Engenharia de Software além da pesquisa sobre *code smells* e refatoração, para avaliar o quão pertinente a gamificação de uma ferramenta voltada para esse tópico (Capítulo 2 e 2.4). A segunda atividade foi a definição dos requisitos do sistema. O enfoque dessa atividade foi considerar o sistema como um todo, ou seja, nessa atividade o sistema gamificado de forma geral é analisado (Capítulo 3.1).

A terceira atividade teve como objetivo projetar a estrutura do sistema, ou seja, selecionar as tecnologias e modelar a arquitetura do mesmo (Capítulo 3.2). Já a quarta atividade caracteriza a implementação do CleanGame (Capítulo 3.3).

Na segunda fase do projeto, na quinta atividade, foi realizado um experimento com estudantes utilizando o sistema, para avaliar o quão significativo e pertinente é o seu uso (Capítulo 4), e por fim, na sexta atividade, os resultados do mesmo foram tabulados e analisados, tendo como objetivo a avaliação da aceitabilidade e eficiência do CleanGame (Capítulo 4.7).

As atividades 1 e 2 contribuem diretamente para os objetivos específicos OE_1 (Investigar o uso de gamificação em ferramentas e sistemas para apoio ao ensino de Engenharia de Software) e OE_2 (Avaliar ferramentas de identificação de *code smells*). As etapas 3 e 4 contribuem para o objetivo específico OE_3 (Implementar o sistema utilizando os conceitos fundamentais da gamificação). Por fim as atividades 5 e 6 contribuem para o objetivo específico OE_4 (Avaliar o uso do sistema desenvolvido através de um experimento).

1.4 Contribuições e Relevância

De acordo com o nosso conhecimento, a ferramenta apresentada é a primeira plataforma educacional a realizar uma abordagem de reforço pós-treinamento[2] para identificação de *code smells*. Para corroborar os benefícios da abordagem gamificada, foram realizadas duas avaliações: um experimento envolvendo 18 participantes e uma pesquisa de atitude, realizada após o experimento. Dessa forma, as principais contribuições dessa pesquisa são três:

1. Apresentação do CleanGame: uma plataforma gamificada para reforço pós-treinamento dos conceitos e habilidades de identificação de *code smells*.

2. De acordo com as evidências atuais, argumenta-se que um ambiente gamificado é mais eficaz para transmitir habilidades de identificação de *code smells*, mantendo os alunos mais envolvidos do que uma abordagem tradicional (ou seja, orientada por IDE). Por isso, um experimento é realizado para investigar o impacto e a solidez da gamificação para apoiar e envolver os alunos durante as atividades de identificação de *code smells*.

[2] Uma série de pequenas lições ou atividades de aprendizado que apoiam um conceito ou habilidade central.

3. Uma pesquisa de atitude foi administrada aos participantes do experimento para obter uma visão geral de suas atitudes em relação ao CleanGame e as vantagens e desvantagens de usar uma abordagem gamificada para a identificação de *smells*.

1.5 Publicações e Premiações

Durante o desenvolvimento deste projeto de pesquisa um artigo foi publicado no *33º Brazilian Symposium on Software Engineering* (SBES), sendo inclusive premiado como primeiro melhor artigo da trilha de educação do SBES.

- Hoyama Maria dos Santos, Vinicius H. S. Durelli, Maurício R. de A. Souza, Eduardo Figueiredo, Lucas Timoteo da Silva, Rafael Serapilha Durelli: **CleanGame: Gamifying the Identification of Code Smells.** SBES 2019: 437-446

1.6 Organização do Trabalho

Além desse capítulo este trabalho está organizado da segunte maneira: O Capítulo 2 descreve a fundamentação teórica sobre *code smells* e refatoração, educação em Engenharia de Software e gamificação. Em seguida o Capítulo 2.4 apresenta os trabalhos relacionados com este trabalho. O Capítulo 3 apresenta o CleanGame, em termos de estrutura e implementação. O Capítulo 4 descreve a avaliação do CleanGame, e por fim o Capítulo 5 apresenta a conclusão deste trabalho de pesquisa.

Capítulo 2 apresenta os principais conceitos para elaboração e execução deste trabalho; nele são expostos os conceitos sobre refatoração e *code smells*, além de exibir algumas ferramentas e suas características para identificação dos mesmos. Também são explorados temas como Educação em Engenharia de Software e Gamificação. Por fim, apresenta alguns trabalhos relacionados com gamificação e educação em Engenharia de Software, que também serviram para nortear as diretrizes seguidas para realização deste trabalho.

Capítulo 3 apresenta as diretrizes seguidas para o desenvolvimento do sistema gamificado, seu funcionamento e interfaces; nesse capítulo são mencionados itens como: seleção dos elementos utilizados, metodologia de desenvolvimento, bem como a forma de integração da ferramenta de detecção de *code smells* com o sistema, seus perfis de usuários e módulos.

Capítulo 4 apresenta os resultados do experimento realizado para validar o CleanGame. O experimento envolveu 18 alunos do curso de ciência da computação da Universidade Federal de Minas Gerais.

Capítulo 5 apresenta a conclusão deste projeto, resumindo os resultados em relação aos objetivos. São apresentadas as contribuições e propostos os trabalhos futuros.

2 BACKGROUND

Este capítulo apresenta os tópicos que fundamentam e norteiam esta pesquisa, sendo relevantes para a compreensão da mesma. Vislumbrando este propósito, aspectos como conceito e importância da identificação dos *code smells* são discutidos na Seção 2.1.1. Na Seção 2.1.2 os principais conceitos sobre refatoração são descritos. A Seção 2.1.3 apresenta um conjunto de apoio computacional para auxiliar a identificação de *code smells*. Os conceitos sobre Educação e sua atual situação no que diz respeito a Engenharia de Software é discutido na Seção 2.2. Finalmente, a descrição sobre o que é gamificação e seu propósito é elucidado na Seção 2.3.

2.1 Identificação de *Code Smells* e Refatoração

A manutenção de software é essencial para manter e adequar o mesmo durante seu ciclo de vida (ALVES; MARTINS; PAULISTA, 2017). Estudos apontam que o esforço desprendido para tal tarefa é quase mais da metade do esforço total investido em qualquer sistema de software durante sua vida útil (REHMAN et al., 2018; MANSOOR et al., 2017).

A qualidade de software é crítica e essencial em diferentes tipos de sistemas e sua baixa qualidade pode levar a situações delicadas (AL-QUTAISH, 2010). O enfoque a estudos relacionados à qualidade de software pôde ser observado nas últimas décadas, já que houve uma maior integração de softwares com atividades cotidianas (PRESSMAN, 2011).

Dessa forma, a estrutura interna de códigos afeta na qualidade do mesmo, já que, mesmo não sendo necessariamente uma causa direta de falha, pode colaborar para a inserção de erros responsáveis por futuras falhas (SOUZA et al., 2017). Com isso, é necessário obter formas de melhorar a qualidade de um código existente ou ainda evitar a inserção desses problemas logo em sua criação.

Uma técnica utilizada para melhorar um código existente é a refatoração (BARROS, 2015) - técnica que proporciona a alteração da estrutura interna do código sem alterar seu funcionamento (FOWLER; BECK, 1999). Porém, para realizar tal ação, é necessário que os desenvolvedores identifiquem os trechos a serem refatorados. Fowler e Beck (1999) sugerem que *code smells* nos códigos são indícios que o mesmo deva ser refatorado, e salientam que, não necessariamente eles provocam um erro no funcionamento do sistema, mas pode atrapalhar na manutenibilidade, legibilidade, entre outros.

Diante disso, pode-se observar que um código onde os *code smells* são identificados e ocorre a refatoração, ou ainda códigos que são criados evitando a inserção de *code smells*

pode se tornar um software com maior qualidade, já que apesar de os *smells* não, necessariamente, causarem defeitos no funcionamento do sistema, podem afetar negativamente em sua qualidade (GARCIA et al., 2009; GUO; SHI; JIANG, 2019; KAUR, 2019).

Com isso, observa-se que o cumprimento e a promoção de boas práticas de desenvolvimento é um mecanismo poderoso para promover a sustentabilidade do software (ARTAZA et al., 2016). As seções que seguem apresentam os conceitos sobre *code smells*(2.1.1) e refatoração(2.1.2) com mais detalhes.

2.1.1 *Code Smells*

Como mencionado, a qualidade do software depende de diversos fatores, entre os aspectos prejudiciais para mesma tem-se os *code smells*, que podem causar prejuízos à qualidade interna e podem inclusive levar à falhas do sistema, quando ignorados (FU; SHEN, 2015).

Apresentados e definidos por Fowler e Beck (1999), a partir de descrições informais, os *code smells*, primeiramente, foram expostos como falhas existentes no código que devem ser removidas através de refatorações (MUNRO, 2005), conceito que será melhor explorado na próxima Seção (Seção 2.1.2).

A presença dos *code smells* em códigos representa violações de boas práticas e princípios de programação, podendo ocasionar um impacto negativo na qualidade do mesmo. Inclusive, classes com *code smells* são um pouco mais relacionadas à ocorrência de *bugs* (NASCIMENTO; SANT'ANNA, 2017).

Os *code smells* também podem ser apresentados como um sintoma notável de dívida técnica (TUFANO et al., 2017), que é definida por Cunningham (1992) como soluções "rápidas" para problemas gerados ao longo do desenvolvimento, e que teoricamente deveriam ser melhoradas em um segundo momento (através da refatoração), o que muitas vezes não é o que ocorre, fazendo dessa forma que a "dívida" só aumente (BROWN et al., 2010).

É importante salientar que os *code smells* não visam fornecer critérios precisos de quando refatorar, mas sim sugerem indicações que algo pode estar errado no *design* ou no código. Portanto, é necessário que os programadores desenvolvam seu próprio senso de quando um indício realmente necessita de mudança (SRIVISUT; MUENCHAISRI, 2007).

O processo de detecção e remoção dos *code smells* não é trivial (SRIVISUT; MUENCHAISRI, 2007). Acredita-se, ainda, que alunos ou profissionais iniciantes tenham uma dificuldade maior para identificá-los, categorizá-los e corrigi-los, já que usualmente é uma ativi-

dade que demanda conhecimento e o processo de identificação e reparo dos mesmos é inexplícito (ITO et al., 2014).

Em um estudo realizado por Tufano et al. (2017), foi investigado quando, por quem e em qual ocasião os *code smells* eram inseridos nos códigos, qual a capacidade de sobrevivência e quando os mesmos eram retirados dos códigos. O estudo apresenta descobertas importantes:

- os artefatos de código são afetados por *code smells* desde sua criação;
- os artefatos de código que adquirem *code smells* ao longo do processo de manutenção apresentam características diferentes dos inseridos na criação dos mesmos;
- implementar novos recursos e aprimorar os existentes são as principais atividades em que desenvolvedores tendem a incluir *code smells*, e ainda existem casos em que a introdução dos mesmos aconteceram em operações de refatoração;
- os desenvolvedores novatos não são, necessariamente, responsáveis pela introdução de *code smells*, enquanto os desenvolvedores com altas cargas de trabalho e pressão para entrega de sistemas, são mais propensos a introduzir instâncias de *code smells*;
- os *code smells* têm uma alta capacidade de sobrevivência e raramente são removidos como consequência direta das atividades de refatoração;

Diante dos itens mencionados, Tufano et al. (2017) relata a importância de evitar a introdução dos *code smells*, colaborando para que práticas de refatoração sejam necessárias com menos frequência, já que essa prática pode por vezes levar a impactos no código.

Com isso, percebe-se a importância de que desenvolvedores saibam identificar *code smells* e consigam, além de removê-los, evitar a inserção dos mesmos, reforçando a valia de uma ferramenta que permita de forma lúdica o ensino desses conceitos.

Na Tabela 2.1 são apresentados alguns *code smells* e suas definições, feitas por Fowler e Beck (1999). Os *code smells* expostos foram escolhidos baseado nos utilizados em experimentos realizados nesse projeto de pesquisa e os mais frequentes em diferentes domínios segundo uma investigação realizada por Fontana et al. (2013):

Tabela 2.1 – Listagem - *Code smells*

Code smell	**Descrição**
God Class[1]	Refere-se àquelas classes que tendem a centralizar a inteligência do sistema, uma instância de uma classe executa a maior parte do trabalho, uma classe que faz coisa demais no sistema.
Feature Envy[1]	Parte do código de uma classe "inveja" outra classe, por exemplo, um método de uma classe usa atributos somente da outra classe.
Divergent Change[1]	Ocorre quando uma classe pode mudar frequentemente de diferentes formas e por razões distintas
Long Method[1]	Um método contém muitas linhas de código, métodos que centralizam a funcionalidade da classe.
Shotgun Surgery[1]	Oposto do Divergent Change, toda vezes que você alterar uma classe, você tem que fazer várias pequenas mudanças em outras classes diferentes .
Duplicate Code	A mesma estrutura de código em mais de um lugar.
Data Class	Uma classe que contém apenas campos e métodos brutos para acessá-los, são simplesmente detentoras de dados usados por outras classes; são classes não contêm nenhuma funcionalidade adicional e não podem operar de forma independente.
Schizophrenic Class	Uma classe que captura duas ou mais abstrações principais, afeta negativamente a capacidade de compreender e alterar de forma isolada as abstrações individuais.

Fonte: Adaptada de Fontana et al. (2013).

Na próxima Seção (2.1.2) o conceito sobre refatoração será apresentado e as refatorações sugeridas para sanar os *code smells* citados são expostas. Como mencionado anteriormente os *code smells* aqui apresentados foram em número reduzido[1], visando expor os utilizados no experimento deste trabalho, e os mais frequentes.

2.1.2 Refatoração

O desenvolvimento de software é um processo não trivial que geralmente se estende por anos. Espera-se que desenvolvedores escrevam códigos rápidos em um tempo mínimo, encontrando as melhores soluções para determinado problema. Com isso, frequentemente o aumento na complexidade do projeto e acúmulo da dívida técnica substancial é notado à medida que o software cresce (SINGH; SINGH, 2017).

Normalmente, os *code smells*, anomalias ou antipadrões, se referem ao estado que afeta o desenvolvimento do software e podem dar indicações de que existem problemas no código.

[1] *Code smells* utilizados no experimento.

[1] Alguns outros *code smells* podem ser encontrados no endereço: https://refactoring.guru/refactoring/smells

Para corrigir esses "defeitos de *design*", operações de refatoração devem ser aplicadas (CHUG; GUPTA, 2017).

A refatoração, proposta por Fowler e Beck (1999), é vista como uma maneira eficaz de melhorar o *design* do código, tanto no momento do desenvolvimento original quanto durante a manutenção do código legado.

Considerada um processo para melhorar a estrutura interna de um sistema sem alterar a funcionalidade do mesmo, a refatoração tem como objetivo melhorar a estrutura, reduzir a complexidade e facilitar o entendimento, e pode ser feita no código-fonte ou no modelo (DHARMAWAN; ROCHIMAH, 2017).

Embora aplicar a refatoração no lugar certo possa ser benéfico para a melhoria incremental da qualidade do software, decidir qual refatoração aplicar e onde aplicá-la é uma tarefa difícil (PAN; JIANG; LI, 2013). Já que, para melhorar a qualidade reduzindo ou removendo os defeitos, muitas vezes a refatoração é feita de forma manual, tendo como principal objetivo a alteração do código com segurança(KAUR; SINGH, 2017).

Alguns ambientes de desenvolvimento populares para uma variedade de linguagens fornecem ferramentas automatizadas de refatoração de códigos, como citado por Murphy-Hill e Black (2008)[2] em seu trabalho, entre elas: Eclipse[3], Microsoft Visual Studio [4], Xcode Xcode [5] e Squeak [6].

Embora existam bastante ferramentas com as técnicas de refatoração implementadas nas mesmas de forma automatizada, como mencionado anteriormente, Fontana et al. (2013) contrapõem sobre essas técnicas em seu estudo, onde verificaram manualmente ocorrências de *code smells* apontados por detectores automatizados e descobriram um grande número de falsos positivos.

Tem-se ainda que, a maioria das classes em que a refatoração é sugerida estão vinculadas com *code smells*. No entanto, um número muito reduzido realmente remove os mesmos, ou seja, é possível que a refatoração apenas tenha mitigado o problema, sem contudo, necessariamente remover completamente o *code smell* (BAVOTA et al., 2015).

Dessa forma, ainda que possa ser dito que a refatoração pode "curar" *code smells* existentes em um código, o conceito de *code smells* é vago e propenso a interpretações subjetivas,

[2] No endereço http://refactoring.com/ é apresentada uma lista de algumas ferramentas
[3] http://eclipse.org
[4] http://msdn.microsoft.com/vstudio
[5] http://developer.apple.com/tools/xcode
[6] http://squeak.org

e mesmo que existam ferramentas de detecção de *smells*, o julgamento humano é indispensável para avaliá-los no contexto do projeto onde são encontrados e determinar se a refatoração deve ser realmente feita (FONTANA; BRAIONE; ZANONI, 2012).

Uma lista abrangente de operações de refatoração sugeridas para remover determinado *smell* é apresentada por Fowler e Beck (1999). Algumas dessas sugestões foram descritas no trabalho desenvolvido por Chug e Gupta (2017) e expostas na Tabela 2.2, vale ressaltar que os *smells* abordados na tabela são baseados nos *code smells* apresentados na seção anterior (Seção 2.1.1):

Tabela 2.2 – Listagem - Refatorações sugeridas para os *Code smells*

Code Smell	**Refatorações Sugeridas**
God Class[2]	*Extract Class:* dividir a classe em duas; *Extract Subclass:* criar uma subclasse para a classe.
Feature Envy[2]	*Move Method* e *Move Field*: mover métodos e atributos entre classes; *Inline Class:* juntar duas classe em uma.
Divergent Change[2]	*Extract Class:* dividir a classe em duas.
Long Method[2]	*Extract Method:* dividir o método em dois; *Replace Method with Method Object:* transformar um método em uma classe.
Shotgun Surgery[2]	*Move Method* e *Move Field:* mover métodos e atributos entre classes; *Inline Class:* juntar duas classes em uma
Duplicate Code	*Extract Method:* dividir o método em dois; *Pull up method*: move métodos de uma classe para uma das suas super classes;
Data Class	*Move Method*: move o método para uma classe que contém a maioria dos dados usados pelo método; *Extract method*: permite extrair uma seção do método em seu próprio método;
Schizophrenic Class	*Extract method*: permite extrair uma seção do método em seu próprio método;

Fonte: Adaptada de Chug e Gupta (2017).

Diante disso, observa-se o quão importante é que os envolvidos no desenvolvimento do código criem senso crítico de quando refatorar, isto é, sejam capazes de identificar *code smells* que sejam nocivos para seus sistemas, para então melhorar sua qualidade.

Dessa forma, ainda que existam ferramentas que identifiquem possíveis refatorações (identifiquem os *code smells*), é esperado que os desenvolvedores consigam determinar como e quando fazer isso.

Na próxima seção(2.1.3) são apresentadas duas ferramentas de detecção automática de *code smells*, e um comparativo entre as mesmas será realizado.

[2] *Code smells* utilizados no experimento.

2.1.3 Ferramentas de Identificação de *Code Smells*

Existem técnicas de identificação de *code smells* que podem ser utilizadas no código-fonte, usando análise manual ou automatizada. As ferramentas que suportam análises automatizadas dependem, geralmente, de diferentes estratégias de detecção (métricas de software, por exemplo) (VIGGIATO; OLIVEIRA; FIGUEIREDO, 2017).

Neste trabalho são exibidas duas ferramentas de detecção de *code smells* e realizada uma análise acerca das mesmas. Essa análise foi realizada com o intuito de selecionar uma para utilização nesse projeto, já que o mesmo necessita de uma ferramenta para detectar possíveis *code smells* em códigos JAVA.

As ferramentas apresentadas foram selecionadas pela capacidade de analisar programas em JAVA, serem instaladas e configuradas a partir de arquivos fornecidos e disponibilizar em sua saída uma lista de ocorrências de *smells*; além de sua popularidade e familiaridade do autor com as mesmas. Ao final do comparativo, a ferramenta selecionada será destacada.

As ferramentas selecionadas para o comparativo e posterior seleção para utilização no projeto, são: PMD[7], um plug-in *open source* que utiliza a análise estática em códigos-fonte; e *JDeodorant*[8], um *plug-in* do Eclipse voltado para identificação de *code smells*.

Abaixo são expostas, em suma, as características de cada ferramenta:

- **PMD:** realiza análise estática em códigos-fonte e, apesar de possuir várias regras implementadas para serem executadas, permite que novas regras sejam criadas para impor práticas de codificação específicas para organização; as mesmas devem ser selecionadas na execução para que o rastreio seja realizado. Como saída, o PMD emite um relatório com os *code smells* selecionados para inspeção, em formato XML, JSON, e HTML.

- **JDeodorant:** voltado para identificação de cinco *code smells* específicos, a saber: *Feature Envy*, *Type Checking*, *Long Method*, *God Class* e *Duplicated Code*. Ele emprega uma variedade de métodos e técnicas para identificar *code smells* e sugere as refatorações apropriadas para resolvê-los. Embora permita a seleção de qual *smell* deverá ser identificado dentre os cinco suportados, não permite criação de regras para análise como disponível no PMD.

[7] http://pmd.sourceforge.net/
[8] http://jdeodorant.com/

As ferramentas mencionadas foram analisadas de acordo com comparativos encontrados na literatura sobre as mesmas e selecionada uma conforme sua pertinência para o trabalho.

Em uma pesquisa realizada por Paiva et al. (2015) foram avaliadas e comparadas três ferramentas de detecção de *code smells* (*inFusion*(iF), *JDeodorant*(JD) e PMD), para isso foram executadas em diferentes versões de um mesmo sistema buscando três *code smells* específicos: *God Class*, *God Method* e *Feature Envy*.

Como resultado obtiveram dados que faziam referência à quantidade de *code smells* encontrados, taxa de precisão e taxa de *recall* por ferramenta. Nesse trabalho são apresentados apenas os dados das ferramentas mencionadas anteriormente[9] (PMD e *JDeodorant*). A Tabela 2.3 traz a adaptação dos resultados extraídos por Paiva et al. (2015).

Tabela 2.3 – Comparativo - PMD x JDeodorant (JD)

	Quantidade		Taxa-Precisão		Taxa-*recall*	
	PMD	JD	PMD	JD	PMD	JD
God Class	8	85	78%	28%	17%	58%
God Method	16	100	100%	35%	26%	50%
Feature Envy	-	69	-	13%	-	48%

Fonte: Adaptada de Paiva et al. (2015).

A partir dos dados apresentados os autores concluíram que embora a ferramenta *JDeodorant* tenha identificado muitos *code smells*, ela gerou relatórios com mais falsos positivos, o que aumenta o esforço de validação. Por outro lado, PMD identificou mais *code smells* válidos, no entanto, não identificaram alguns, e embora reduza o esforço de validação, deixa de destacar possíveis *code smells*.

Um novo estudo foi realizado realizado pelos mesmos autores, desta vez quatro ferramentas foram analisadas (*inFusion*, *JDeodorant*, PMD, e *JSpIRIT*), também, rodando em diferentes versões dos mesmos sistemas de software. Seguindo o mesmo raciocínio do estudo exposto anteriormente, descartaremos as ferramentas *inFusion* e *JSpIRIT*, já que para esse projeto são analisadas apenas a *JDeodorant* e *PMD*. Os *code smells* a serem identificados foram os mesmos do primeiro experimento, a saber: *God Class*, *God Method* e *Feature Envy*.

Como resultado apresentaram os mesmos dados elencados da pesquisa anterior, ou seja, faziam referência à quantidade de *code smells* encontrados, taxa de precisão e taxa de *recall* por

[9] As duas ferramentas direcionadas para análise foram selecionados pela familiaridade da autora com as mesmas.

ferramenta, como pode ser observado na Tabela 2.4, adaptada de Paiva et al. (2017).

Tabela 2.4 – Comparativo 2 - PMD x JDeodorant (JD)

	Quantidade		Taxa-Precisão		Taxa-*Recall*	
	PMD	JD	PMD	JD	PMD	JD
God Class	33	98	36%	8%	100%	70%
God Method	13	599	85%	8%	17%	82%
Feature Envy	-	90	-	0%	-	-

Fonte: Adaptada de Paiva et al. (2017).

A análise realizada pelos autores foi bem similar ao do primeiro trabalho, no qual relataram que para todos os *code smells*, o *JDeodorant* foi o que identificou o maior número, mas relatou muitos falsos positivos, o que significa um aumento no esforço de validação. Já a PMD teve maior precisão, porém algumas classes afetadas não foram relatadas.

A partir dos dados fornecidos, nota-se que ambas as pesquisas trouxeram resultados bem parecidos, podendo concluir que a ferramenta *Jdeodorant* faz uma cobertura melhor dos *code smells* existentes, porém apresenta mais falsos positivos, ao contrário da PMD, que reporta um número bem inferior de *code smells* mas com maior precisão.

No trabalho a ser desenvolvido a escolha pela ferramenta foi baseada em alguns aspectos: possuir maior precisão, ter mais opções de *code smells* para identificação, dar mais liberdade para manipulação, possuir facilidade para integração com outros sistemas e formato do arquivo de retorno com os *smells*. Diante disso, e pensando no esforço para validação dos *code smells* identificados, a escolhida para compor esse projeto foi a PMD.

2.2 Educação e Prática em Engenharia de Software

A área de Engenharia de Software está presente, como disciplina, em praticamente todos os currículos dos cursos da área de computação, sendo possivelmente, uma das áreas mais avançadas no país (ZORZO et al., 2017). Com isso, o desenvolvimento de um currículo na área da computação deve ser sensível às mudanças na tecnologia, aos novos desenvolvimentos em pedagogia e à importância da aprendizagem ao longo da vida (IEEE,ACM, 2016).

Uma das principais características dos cursos de Engenharia de Software é que os alunos devem lidar com problemas reais, semelhantes aos que enfrentarão na indústria, contando ainda

com algumas orientações sobre como lidar com essa realidade com a qual não estão acostumados (BASTARRICA; PEROVICH; SAMARY, 2017).

Espera-se que instituições educacionais adotem estratégias explícitas para responder às mudanças relacionadas ao ensino e tenham sucesso no processo (DEVADIGA, 2017). Como suporte para tal, é possível encontrar nas "Diretrizes Curriculares para Programas de Graduação em Engenharia de Software" (IEEE,ACM, 2016) as competências necessárias e esperadas dos estudantes, já que o documento fornece um conteúdo sobre as habilidades que devem ser trabalhadas no decorrer dos cursos de Engenharia de Software.

Entre as habilidades mencionadas no documento IEEE,ACM (2016), "Preparação para a prática profissional" é uma delas, que muitas vezes não é alcançada com êxito nos cursos. Já que tornar o ensino-aprendizagem mais eficaz através de métodos alternativos de ensino, no sentido de torná-lo menos entediante, tem sido um grande desafio (SAVI; WANGENHEIM; BORGATTO, 2011), embora a busca por inovação nesse sentido venha sendo contínua (BORRAS-GENE; MARTINEZ-NUNEZ; FIDALGO-BLANCO, 2016).

Como forma de maximizar o ensino e tirar o professor como principal agente da aprendizagem por meios tradicionais Savi, Wangenheim e Borgatto (2011), sugere-se a inserção da tecnologia à mesma (SAVI; WANGENHEIM; BORGATTO, 2011), já que com esse tipo de intervenção, espera-se que alunos sejam mais motivados e estimulados a realizar tarefas.

O ensino deve fornece aos alunos o conhecimento necessário para fazer a transição do mundo acadêmico para empresas com estrutura definida. No entanto, a fluidez, o risco, a sensibilidade ao tempo e a incerteza das mesmas exigem um conjunto dinâmico e ágil de habilidades para identificar, conceituar e fornecer recursos rapidamente, conforme as necessidades do mercado (DEVADIGA, 2017).

Com isso, é possível observar que profissionais recém-formados em ciência da computação não estão bem preparados para enfrentar trabalhos industriais (BASTARRICA; PEROVICH; SAMARY, 2017).

Embora as abordagens teóricas sejam bem aplicadas nas universidades, a Engenharia de Software exige mais abordagens práticas, dinâmicas e métodos envolventes. Fazendo-se necessária a participação ativa dos alunos em projetos reais (FERNANDES; WERNER, 2019).

Uma das principais características dos cursos de Engenharia de Software é que os alunos devem lidar com problemas reais semelhantes aos que enfrentarão na indústria, contando ainda com algumas orientações sobre como lidar com essa realidade à qual não estão acostuma-

dos (BASTARRICA; PEROVICH; SAMARY, 2017). No entanto, devido às restrições de tempo e o escopo inerentes ao ambiente acadêmico, existem poucas oportunidades para explorar essas vertentes relacionadas à prática da Engenharia de Software(FERNANDES; WERNER, 2019).

Reafirmando, dessa forma, a importância de expor os alunos a meios alternativos de ensino que simulem a realidade profissional, e os coloque como agentes ativos de sua aprendizagem prática.

2.3 Gamificação

O termo gamificação é relativamente novo, e refere-se ao uso de elementos que proporcionem experiências e motivações semelhantes às dos jogos e, consequentemente, afetem o comportamento do usuário (KOIVISTO; HAMARI, 2019). Ou seja, se refere ao uso de elementos e técnicas de *design* de jogos em contextos não relacionados a jogos, com o objetivo de motivar as pessoas em uma variedade de tarefas e induzi-las a comportamentos, bem como melhorar sua motivação e envolvimento em uma tarefa específica (DETERDING et al., 2011a).

Avanços na tecnologia em diversas áreas estimularam pesquisas ativas sobre gamificação de processos e atividades em áreas "sérias", além da educação, tais como: treinamento corporativo, gestão, administração, marketing entre outros (USKOV; SEKAR, 2014a); visando melhorias nos processos, já que tal prática propõe o engajamento dos usuários.

É possível observar continuidade em sua popularidade, desde o início de 2010, tanto no âmbito educacional como na indústria, mas a educação, em especial, tem recebido uma atenção significativa (MAJURI; KOIVISTO; HAMARI, 2018; BORGES et al., 2014). Uma evolução positiva nos últimos anos tem acontecido na área de Engenharia de Software. Inclusive, nas últimas décadas, a melhoria dos processos de software usando padrões tem ganhado um foco especial (GARCÍA et al., 2017).

2.3.1 Elementos de Jogo

Embora alguns elementos não sejam encontrados em todos os jogos, estão presentes na maioria e são fundamentais para a composição de um sistema gamificado, como mencionado por Deterding et al. (2011a). Eles apresentam, também, alguns padrões de *design* de interface (Tabela 2.5) que podem ser úteis para sistemas gamificados.

Tabela 2.5 – Níveis de elementos de *design* de jogo de Deterding et al. (2011a)

Nível	Descrição	Exemplo
Padrões de *design* da interface do jogo	Componentes comuns e bem-sucedidos de *design* de interação e soluções de *design* para um problema conhecido em um contexto, incluindo implementações prototípicas	Crachás, tabelas de classificação, níveis
Padrões de *design* de jogos e mecânica	Comumente recorrendo partes do *design* de um jogo que dizem respeito ao jogo	Restrição de tempo, recursos limitados, regresso
Princípios e heurísticas do *design* de jogos	Diretrizes de avaliação para abordar um problema de *design* ou analisar uma determinada solução de *design*	Jogo duradouro, objetivos claros, variedade de estilos de jogo
Modelos de jogo	Modelos conceituais dos componentes de jogos ou experiência de jogo	Mecânica - Dinâmica - Estética (MDA); desafio, imaginação, curiosidade; partes de *design* de jogos; Elementos Essenciais da Experiência de Jogo (CEGE)
Métodos de *design* de jogos	Práticas e processos específicos de *design* de jogos	Playtesting, *design* playcentric, *design* de jogo conhecido

Fonte: Adaptada de Deterding et al. (2011a).

Como exposto na Tabela 2.5, a gamificação deve possuir características semelhantes a jogos, com propósito de gerar sentimentos aos usuários (desafio, imaginação, curiosidade) e com isso sentirem-se motivados em continuar a atividade proposta.

No *design* de jogos são apresentados elementos que são definidos dentro da gamificação como: Pontos, Níveis, *Badges*, Tabelas de Classificação e *Feedback*(BOAS et al., 2016). Esses elementos são frequentemente considerados na gamificação, por serem utilizados em muitas implementações como recurso (KOIVISTO; HAMARI, 2019). Já que, ao contrário dos jogos, que necessitam de todas as condições necessárias e suficientes para ser um jogo, os sistemas gamificados usam apenas os seus elementos de *design* (DETERDING et al., 2011a).

Complementando, Khaleel et al. (2016) cita com base em estudos anteriores, três elementos de gamificação: mecânica de jogo (painel e barra de progresso), *design* do jogo (usando um distintivo como recompensa) e técnicas de jogo (classificações mostrando pontuações ou marcas), esses que seriam aplicados em um ambiente de aprendizado para projetar uma apresentação de conteúdo de aprendizagem agradável, motivadora e eficaz.

Como mencionado anteriormente, definindo gamificação de maneira ampla, pode-se dizer que o uso de elementos de jogos em contextos não jogos buscam instigar experiências semelhantes a jogos, motivando e tornando a utilização agradável (MORSCHHEUSER et al., 2017). Para explicitar melhor essa definição a mesma será descompactada em *(i)* jogos, *(ii)* elementos, *(iii*) contexto não-jogos e *(iv) design*, de acordo com Deterding et al. (2011b):

(i) Jogos: são caracterizados por regras e competição ou conflitos em relação a resultados específicos ou discretos por participantes humanos de maneira divertida;

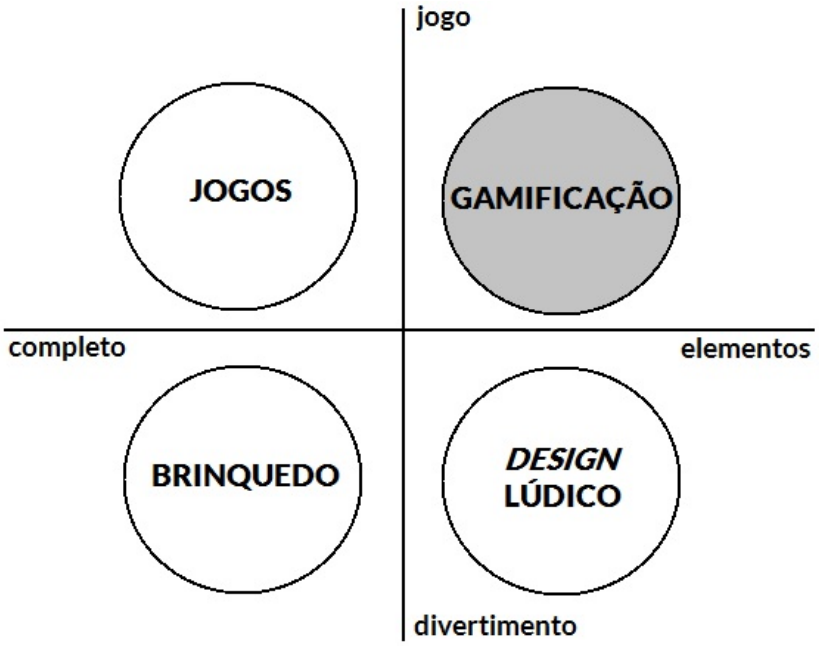
(ii) Elementos: em um "conjunto liberal" , ou seja, qualquer elemento encontrado em qualquer jogo - seriam ilimitados. Já em um "conjunto restrito", em que seriam considerados somente elementos exclusivos dos jogos - seria muito restritivo se não estivesse vazio. Sugere-se então limitar a gamificação para a descrição de elementos que são característicos dos jogos;

(iii) Contexto não-jogos: uso de jogos para fins diferente do seu uso normal, esperado para entretenimento. Não se deve limitar o termo gamificação a contextos, propósitos ou cenários específicos de uso, observando que a alegria de uso, o engajamento ou, em geral, a melhoria da experiência do usuário atualmente servem como contextos de uso popular;

(iv) design: espera-se possuir níveis ou tabelas de classificação, padrões de *design* ou mecânica de jogos; modelos que desafiem e instiguem a curiosidade;

Com a intenção de explicitar, mais detalhadamente, onde a gamificação situa no contexto dos jogos Deterding et al. (2011b) propôs um esquema, a adaptação do mesmo pode ser observada na Figura 2.1, onde é possível observar que a gamificação utiliza elementos de *design* de jogos, porém não possui cunho de entretenimento, ou seja, a gamificação não apresenta características de um jogo completo, mas sim, faz uso dos seus elementos com o objetivo que o "jogador" se estimule através de todo o cenário montado envolto aos elementos dos jogos.

Figura 2.1 – Gamificação no jogo e brincadeira, completo e elementos



Fonte: Adaptado de Deterding et al. (2011b)

Uma abordagem formal para entender melhor os jogos, ou ainda a ponte entre o desenvolvimento do jogo e o *design* do jogo é o MDA (*Mechanics Dynamics and Aesthetic*), um *framework* de *game design* ou gamificação, baseado na teoria do *design* de jogos (DETERDING et al., 2011a). Ele consiste em três conceitos (mecânica, dinâmica e estética) que se relacionam entre o *designer* de jogos e os usuários do jogo (KIM; LEE, 2015):

(i)Mecânica: descreve os componentes particulares/funcionais do jogo (pontos, níveis, desafios);

(ii)Dinâmica: descreve o comportamento em tempo de execução do jogo;

(iii)Estética: descreve as respostas emocionais desejáveis evocadas no jogador.

Com esses conceitos é possível dividir a gamificação em três categorias principais e trabalhar em cada categoria, facilitando o resultado final de um sistema gamificado.

Para exemplificar a gamificação, pode-se mencionar alguns usos da mesma em contextos variados, como no *Foursquare*[10] que é uma rede social móvel baseada em geolocalização, que pontua os jogadores dependendo da quantidade de *check-ins* que ele faz em determinado local (ALVES et al., 2012); o Duolingo[11], que é uma plataforma colaborativa que utiliza ele-

[10] https://pt.foursquare.com/
[11] https://pt.duolingo.com/

mentos gamificados para o ensino de línguas (SOUZA; SOUTO, 2015); ou o Kahoot[12], que é uma plataforma de ensino gratuita que possibilita a criação de quatro tipos de atividades *online*: *Quizzes*, *Discussion*, *Jumble*, e *Survey*.

2.3.2 Gamificação no Ensino de Engenharia de Software

A gamificação tem ganhado espaço entre pesquisadores e educadores na formação em Engenharia de Software no processo de ensino-aprendizagem, para envolver alunos na execução de tarefas específicas ou incorporando determinados comportamentos, mas embora o uso de métodos relacionados a jogos não seja novidade, a gamificação é uma tendência recente(SOUZA et al., 2018).

Embora essa introdução esteja em sua infância e dados empíricos ainda sejam necessários, seu potencial é baseado na hipótese de que ela apoia e motiva os estudantes, podendo dessa forma levar a processos e resultados aprimorados de aprendizagem (ALHAMMAD; MORENO, 2018).

Em seu estudo Souza et al. (2018) relatam que no contexto do ensino de Engenharia de Software, o uso da gamificação não é uma ferramenta educacional "autônoma", mas de suporte para motivar os alunos a se adaptarem aos comportamentos desejados e que existe uma falta de padronização em seus objetivos de aprendizagem e classificação de métodos relacionados ao jogo.

Conforme apontado por Wangenheim, Kochanski e Savi (2009), apesar de os indícios de que intervenções com jogos educacionais no ensino de Engenharia de Software, por exemplo, sejam benéficos para aprendizagem, ainda faltam demonstrações formais desses ganhos, necessitando que pesquisadores façam pesquisas e avaliações mais rigorosas quanto sua valia.

Dessa forma esse trabalho tem, também, como intuito contribuir para o estado da arte relacionado a eficácia da gamificação como forma de auxílio na educação em Engenharia de Software.

2.4 Trabalhos Relacionados

Foram considerados trabalhos relacionados à esta dissertação, trabalhos referentes à gamificação e educação em Engenharia de Software encontrados na literatura. Sendo assim, foram

[12] https://kahoot.com/

considerados escopos que envolvam: mapeamentos sistemáticos, relação da gamificação e motivação, desenvolvimento e estratégias de aprendizado e propostas de ferramentas.

Pesquisas indicam que existem efeitos significativos na aprendizagem baseada em jogos refletindo tanto na motivação quanto no desempenho. Diante disso, Su (2016) aborda em seu estudo o impacto que a gamificação pode ter nesses aspectos. Relata que o uso correto da gamificação pode aumentar a motivação e diminuir a ansiedade e carga cognitiva[13], aumentando dessa forma o desempenho acadêmico. Apresenta que pesquisas anteriores tem focado na usabilidade e extensão, porém com menos enfoque nos pontos elencados por ele, e conclui, como mencionado anteriormente, que o uso correto da gamificação pode aumentar a motivação por um lado e diminuir a ansiedade de aprendizagem e carga cognitiva, aumentando dessa forma o desempenho acadêmico.

Um sistema gamificado bem projeto é de grande importância para que o mesmo obtenha êxito em seus resultados e ter ciência de um método para o desenvolvimento do mesmo é relevante (MORSCHHEUSER et al., 2018). Na investigação realizado por Sasso et al. (2017), é apresentada uma estrutura conceitual de como aplicar técnicas de gamificação no domínio da Engenharia de Software, ilustrando essa inserção em dois contextos e discutindo suas descobertas. Esboçam ainda uma proposta sobre como esses sistemas podem ser avaliados.

Em sua pesquisa Anderson, Nash e McCauley (2015) discutem as estratégias de aprendizado adotadas em um ambiente de aprendizado disponível na nuvem, o *Learn2Mine*, que facilita o progresso dos alunos à medida que resolvam problemas de programação. O sistema foi introduzido inicialmente em um curso introdutório e testado em piloto para usabilidade e eficácia e obteve opiniões positivas sobre os pontos elencados. Em um segundo momento, o *Learn2Mine* foi avaliado em um curso de mineração de dados de nível superior, comparando as taxas de envio de alunos e a quantidade de programação realizada para um grupo com acesso à ferramenta *versus* um sem acesso. Como resultado, obtiveram um desempenho significativamente melhor com o uso do *Learn2Mine*.

Já em um estudo realizado por García et al. (2017), além de apresentarem uma estrutura para a aplicação da gamificação em ambientes de Engenharia de Software, foi proposto e aplicado em uma empresa real o uso de um framework para gamificar as áreas de gerenciamento de projetos, gerenciamento de requisitos e testes, do qual se obteve um *feedback* positivo da

[13] Nível de utilização de recursos psicológicos como memórias, atenção, percepção, representação de conhecimento, raciocínio e criatividade na resolução de problemas.

introdução destas mecânicas e práticas no local de trabalho, tendo vários relatos favoráveis da inserção da gamificação nos processos.

No trabalho apresentado por Fu e Clarke (2016) foram descritos os mecanismos de *design* de gamificação usados em um ambiente de aprendizado habilitado para teste de software, denominado *WReSTT-CyLE* (Repositório baseado na Web de Tutoriais de Teste de Software - um ambiente de *Cyberlearning*). Caracterizada uma ferramenta de aprendizado *on-line* altamente integrada, além da tecnologia de gamificação, o *WReSTT-CyLE* sintetiza vários conceitos-chave, incluindo interação social, aprendizado colaborativo e objetos de aprendizado para apoiar a aprendizagem do aluno de forma eficaz e eficiente. Como resultado obtiveram uma relação significativa entre o envolvimento e a motivação dos alunos usando a gamificação e um ambiente de aprendizagem *on-line*, especificamente na área de teste de software.

Kosa et al. (2016) apresentam uma revisão sistemática sobre jogos em educação de Engenharia de Software, relatam que a noção de jogo é usada tanto para jogar quanto para projetar, em abordagens de jogos digitais, não digitais ou ambos; mencionam ainda o resultado positivo do uso dos mesmos na aprendizagem, e apontam a necessidade de mais estudo empíricos, já que ainda não existem muitos estudos que tentam desenvolver diretrizes de *design* ou heurísticas para o desenvolvimento de jogos para educação em Engenharia de Software, entretanto um estudo selecionado na revisão realizada, menciona descobertas nesse ponto.

É possível observar a gamificação como agente motivador na aprendizagem de Engenharia de Software no trabalho desenvolvido por Poffo et al. (2017), em seu experimento foram usados a maioria dos elementos da gamificação (pontuações, tabela de ranking, desafios, emblemas, além de promover a competição); alunos e professor são usuários. O professor é responsável por cadastrar alunos/atividades e liberá-los; os alunos, por vez, fazem o uso do mesmo. Como resultado tiveram um retorno positivo quanto a motivação e relevância para o aprendizado, além da maior parte dos envolvidos terem considerado um ambiente divertido.

Objetivando envolver os alunos em um curso de técnicas de previsão, melhorar os resultados de aprendizado e aprimorar as habilidades de previsão, Legaki et al. (2019) projetaram uma ferramenta de ensino complementar no contexto do curso de técnicas de previsão usando gamificação. Para a realização do experimento foram selecionados alunos que possuíam o mesmo histórico, e embora cientes de um incentivo para quem participasse de tal experimento a participação era opcional. Como resultado, apesar de algumas limitações mencionadas pe-

los autores sobre a realização do experimento, foi exposto que a gamificação pode influenciar positivamente os resultados de aprendizagem.

O protótipo de um sistema para detecção de mutantes equivalentes é apresentado por Laurent et al. (2017), onde relatam a possibilidade do mesmo ser usado como uma ferramenta independente para desenvolvedores e equipes de testes, mas ressaltam que a ideia é que o mesmo seja usado em uma plataforma de *crowdsourcing* para avaliar os vários parâmetros envolvidos na detecção de mutantes equivalentes, como especialização (codificação e teste), familiaridade com a base de código, complexidade do código e testes. No sistema é permitido que os jogadores estudem mutantes e os matem escrevendo um teste ou rotulando-os como equivalentes. Os jogadores ganham pontos quando matam um mutante ou quando eles concordam com a satisfação de um mutante.

Continuando no contexto do ensino de Engenharia de Software e códigos mutantes, o *Code Defenders* desenvolvido por Rojas e Fraser (2016) é um jogo de teste de mutação baseado em classes Java e testes *JUnit*, que funciona com dois jogadores envolvidos onde: um atacante e um defensor competem uns contra os outros, respectivamente, atacando e defendendo uma classe Java em teste (CUT) e sua suíte de testes.

Em um estudo sistemático realizado por (SOUZA et al., 2018) foram apresentados trabalhos relacionados ao uso de jogos na educação em Engenharia de Software, na Tabela 2.6 é exposto um compilado dos trabalhos relatados por eles. Vale ressaltar que no estudo foram apresentados trabalhos relacionados a abordagens de "Aprendizagem baseadas em jogos", "Aprendizagem baseada no desenvolvimento em jogos"e "Gamificação", aqui são apresentados apenas os relacionados à gamificação.

Tabela 2.6 – Trabalhos relacionados

Sala de aula[1]		**Específicas de ES**[2]	
Ref.	Descrição	Ref.	Descrição
(BERKLING; THOMAS, 2013)	Descreve a configuração de uma sala de aula gamificada para o assunto de ES. Uma plataforma de software foi desenvolvida para apoiar elementos de jogos em sala de aula. Foi considerada um fracasso e reavaliada em (THOMAS; BERKLING, 2013), mantendo os elementos mas não enfatizando-os.	(AKPOLAT; SLANY, 2014)	Descreve uma abordagem para adicionar gamificação ao processo de desenvolvimento de software. Os alunos devem aprender e utilizar técnicas extremas de programação enquanto desenvolve um sistema de software.
(USKOV; SEKAR, 2014b)	Propõe a incorporação de mais de 20 elementos de gamificação em cursos de ES.	(BUISMAN; EEKELEN, 2014)	Propõe a gamificação para aumentar a utilização de ferramentas durante o desenvolvimento. Usaram um plugin para a ferramenta de gerenciamento.
(LASKOWSKI, 2015)	Descreve um experimento implementando algumas das técnicas de gamificação nos cursos de Engenharia de Software e arquitetura orientada a serviços, para verificar se a mesma é aplicável a diferentes cursos no ensino superior e quais são os resultados de tal aplicação	(SINGER; SCHNEIDER, 2012)	Apresenta resultados preliminares de um experimento que incentiva os estudantes de um curso de projeto de software a realizar *commits* mais frequentes ao sistema de controle de versão.
(QU et al., 2014)	Descreve os objetivos do uso da gamificação para treinamento de pessoal na área de Engenharia de Software.	(BELL; SHETH; KAISER, 2011)	Propõe que os alunos realizem testes de software usando um ambiente de jogo (HALO - Highly Addictive, Socially Optimized), com a pretenção de apresentar o teste de software disfarçado.

Fonte: Adaptada de Souza et al. (2018).

Por fim, são apresentados na Tabela 2.8 os trabalhos relacionados à gamificação e sua natureza.

[1] Experiência em sala de aula - refere-se ao uso de elementos do jogo para envolver e motivar os alunos na realização de atividades de aprendizagem.

[2] Gamificação de atividades específicas de Engenharia de Software - aplicação de elementos de jogos para motivar os alunos a praticar habilidades específicas ou executar práticas específicas.

Tabela 2.8 – Trabalhos relacionados e sua natureza

Temas	**Fontes / Referências**
Mapeamento Sistemático	(SASSO et al., 2017), (KOSA et al., 2016).
Relação Gamificação e Motivação	(SU, 2016), (ANDERSON; NASH; MCCAULEY, 2015), (FU; CLARKE, 2016), (POFFO et al., 2017), (DUBOIS; TAMBURRELLI, 2013).
Desenvolvimento e Estratégias de Aprendizado	(GARCíA et al., 2017), (MORSCHHEUSER et al., 2018).
Proposta de ferramenta	(LEGAKI et al., 2019), (LAURENT et al., 2017), (ROJAS; FRASER, 2016), (RAAB, 2012).

Fonte: Do autor (2019).

2.5 Discussão da Literatura

Através do estudo sobre *code smells* e refatoração, observou-se a dificuldade em identificar e aprender sobre tal anomalia e posteriormente aplicar a refatoração. Diante disso, uma investigação e análise sobre o estado da arte do uso da gamificação no ensino em Engenharia de Software foi realizada. Com isso, foi possível identificar as lacunas na literatura e direcionar o desenvolvimento desse trabalho através do embasamento teórico.

Durante o estudo a respeito dos trabalhos relacionados, além de conseguir tirar referências para o desenvolvimento do sistema gamificado, verificou-se que não havia nenhum trabalho semelhante já apresentado.

Dessa forma, observa-se que, embora alguns seguimentos de Engenharia de Software já tenham sidos contemplados com estudos relacionando gamificação ao seu aprendizado, não foi encontrado na literatura, o uso de gamificação para auxiliar especialmente no aprendizado de identificação de *code smells*, como abordado nesse trabalho, principalmente permitindo a integração com códigos reais do *GitHub*.

2.6 Considerações Finais

Este Capítulo apresentou o *background* que embasou o desenvolvimento desse trabalho. O referêncial teórico, trabalhos relacionados e a discussão sobre a pertinência do desenvolvimento do mesmo foram expostos. Com esses pontos discutidos foi possível dar sequencia ao trabalho. No próximo Capítulo 3 o CleanGame é apresentado.

3 CLEANGAME

Este capítulo apresenta o CleanGame[1], um sistema gamificado destinado a colaborar na fixação dos conceitos e na detecção de *smells*. De acordo com Koivisto e Hamari (2019), a gamificação refere-se ao uso de elementos que proporcionem experiências e motivações semelhantes as dos jogos e, consequentemente, afetem o comportamento do usuário.

O CleanGame pretende fornecer aos usuários formas diferentes de fixar os conceitos sobre *code smells*. Sua estrutura baseia-se nos estudos realizados entorno da bibliografia encontrada sobre gamificação de sistemas, *code smells* e educação em Engenharia de Software, que foram apresentados no Capítulo 2.

Neste Capítulo, o CleanGame será detalhado em termos de organização da estrutura, elementos utilizados, funcionalidade, interface e características do sistema em geral. O restante deste capítulo é organizado da seguinte forma: A Seção 3.1 descreve a visão abstrata do sistema contendo os objetivos e escopo do mesmo, além de apresentar seu público-alvo e organização dos componentes. A Seção 3.2 apresenta elementos de jogos utilizados com suas respectivas justificativas de uso. Na Seção 3.3 o sistema é retratado de forma concreta, descrevendo a utilização de ferramentas externas e perfis de usuários. Por fim, os módulos do sistema são apresentados na Seção 3.4.

3.1 Visão Geral da Ferramenta

O objetivo do CleanGame é apoiar estudantes e educadores no treinamento e fixação de conteúdos relacionados a *code smells*, além de poder ser utilizado por profissionais de Tecnologia da Informação (TI). A ideia central é que os alunos possam ser agentes ativos em sua aprendizagem e se deparem com situações reais durante a fixação do conteúdo.

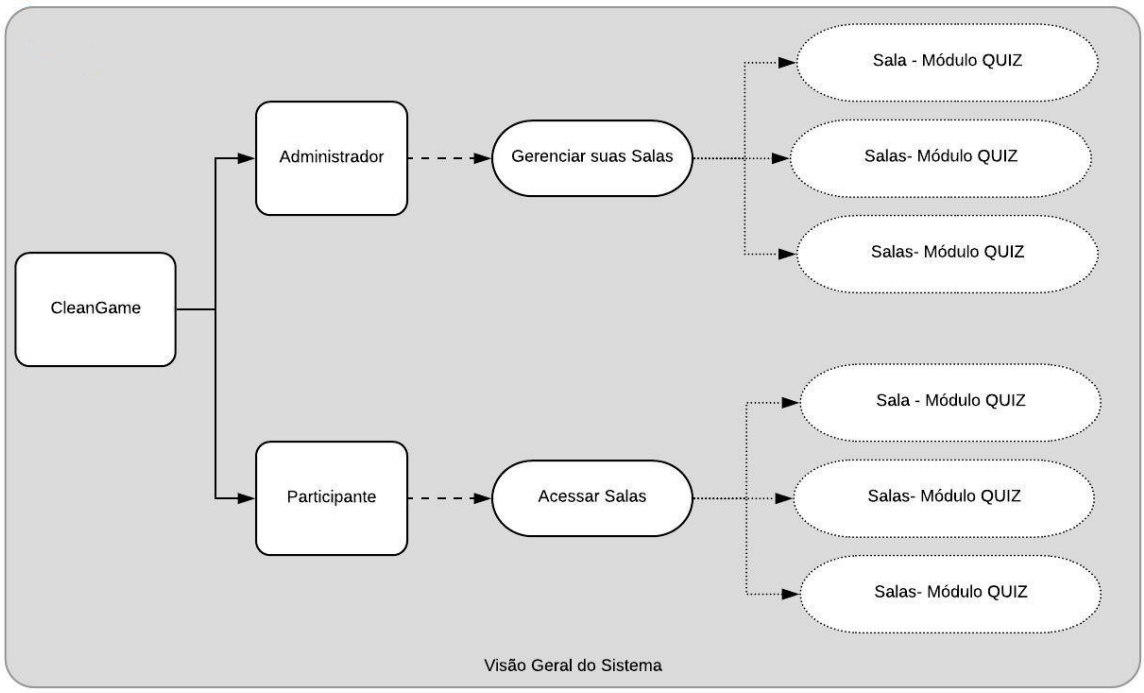
Dessa forma, o público alvo do CleanGame são educadores e educandos, além de ter a possibilidade da inserção em ambientes profissionais. No entanto, o escopo deste trabalho esta limitado ao contexto educacional. A ferramenta é proposta como método alternativo e de suporte na fixação e aprendizagem do conteúdo. Com isso, se espera que os aprendizes se desafiem e se sintam motivados a resolver as tarefas apresentadas.

Conforme a Figura 3.1, o sistema possui dois níveis de usuários, sã eles: *Administrador* e *Participante*, no qual o primeiro pode cadastrar salas e o último participar de salas cadastradas.

[1] Disponível em https://bit.ly/2W6xClB

As salas são instâncias da aplicação que podem adotar um dos três módulos de jogo. *Módulo Quiz* - são apresentadas perguntas e respostas; *Módulo Identificação* - um código real é apresnetado para identificação do *smell*; e *Módulo Batalha* - duelo em que os jogadores inserem *smells* em um código para que seu adversário o encontre. Mais detalhes sobre cada tipo de perfil e dos módulos serão exibidos nas Seções 3.3 e 3.4, respectivamente.

Figura 3.1 – Visão Geral



Fonte: Do autor (2019).

3.2 Elementos de Jogos Usados com Base na MDA

A gamificação do CleanGame, foi planejada conforme modelo MDA (3.2). Estabelecer diretrizes de desenvolvimento e assimilar os conceitos sobre gamificação é de suma importância para que um sistema gamificado possa ser bem aceito pelos usuários, já que a implementação não adequada pode levar ao fracasso do mesmo (ALHAMMAD; MORENO, 2018).

Dessa forma, a ideia é que o CleanGame seja intuitivo e motivador para o usuário, almejando que o mesmo não tenha gasto cognitivo[2] para aprender como manuseá-lo e se sinta

[2] Processo ou faculdade de adquirir um conhecimento.

Figura 3.2 – Sistema Gamificado - Conceitos MDA

Fonte: Do autor (2019).

motivado em utilizá-lo. Para isso, alguns pontos foram definidos para o planejamento do sistema, objetivando a harmonização dos três conceitos do MDA:

1. há um sistema de pontuação para incentivar tanto o jogador avançado quanto o iniciante, possibilitando a competição entre os dois tipos de jogadores;

2. os desafios são organizados em módulos independentes (*QUIZ, IDENTIFICAÇÃO E BATALHA*), de forma que o jogador tenha liberdade de escolher seu tipo de jogo, para que o iniciante não se frustre pela dificuldade e o jogador avançado não ache o mesmo cansativo;

3. o sistema possui um *ranking* para instigar os usuários a melhorar seu desempenho nas atividades;

4. de acordo com seu desempenho os competidores podem receber pontos extras, influenciando de forma direta em sua colocação no *ranking*;

Com isso, a mecânica do sistema foi definida, os elementos citados foram os mais encontrados e aceitos na literatura (Seção 2.3). A Tabela 3.1 apresenta os elementos de jogo que compõe o CleanGame.

Tabela 3.1 – Mecânica - CleanGame

Elemento	**Descrição**
Módulos de jogo	Quizz, Identificação e Batalha.
Pontos	Acumulados a cada desafio, indicando a pontuação e lugar no *ranking* por sala.
Barra de Progresso	Mostra em qual estágio do desafio o usuário está.
Tabela de Classificação	Forma de ranqueamento dos participantes por sala.
Pressão de tempo	Tempo gasto para realizar uma tarefa do desafio.
Dicas	Disponibilizado ao jogador para ajudar em determinado desafio (até três dicas).
Pular desafio	Permite que o jogador pule determinada questão do desafio.
Cooperação	Compartilhamento de uma sala para N indivíduos simultaneamente.
Socialização	Possibilita conversação entre os jogadores de uma sala compartilhada.
Acúmulo de Pontos	Somatório de pontos adquiridos no decorrer das tarefas de determinado desafio.

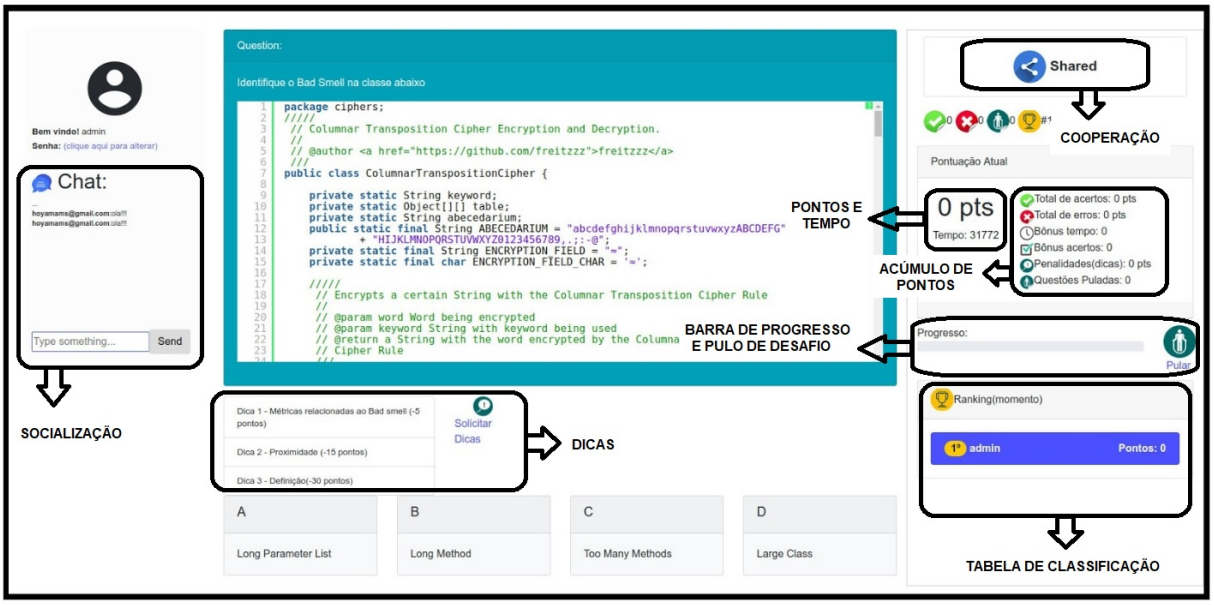
Fonte: Do autor (2019).

As regras do sistema foram criadas para, além de delimitar alguns comportamentos dos usuários, permitir que os mesmos executem ações que os mantenha interessados. Da mesma forma que os elementos, a mecânica aqui mencionada, pode ser utilizada em diferentes seguimentos com o intuito de gamificar um sistema.

Um sistema gamificado tem como propósito transmitir informações aos usuários de forma interativa, a ideia é que eles apresentem elementos de jogos, a fim de atrair a atenção do usuário enquanto ele faz uma tarefa ou aprende um conteúdo que não o atraia em sua totalidade. Na Figura 3.3 é possível observar como cada elemento citado foi utilizado, em sequencia a justificativa do uso de cada um é mencionada (elementos citados na Tabela: 3.1).

- ***Módulos de jogo*:** Os desafios foram pensados de forma que todos os usuários se interessassem de maneira prazerosa por eles, com isso o sistema conta com três categorias de desafios; o sistema possui os módulos QUIZ, IDENTIFICAÇÃO e BATALHA e cada um desses módulos possui tarefas consideradas fáceis, moderadas e difíceis. Seguindo a expectativa de atrair os usuários, esses módulos foram elaborados para que usuários de diferentes níveis de conhecimento se sentissem motivados a realizar as tarefas, não deixando entediado um usuário avançado, nem desmotivado um iniciante, além de permitir a fixação e aprendizado do conteúdo de diferentes maneiras; a forma de apresentação e inserção de conteúdo módulos será detalhada na Seção 3.4.

Figura 3.3 – Elementos Utilizados



Fonte: Do autor (2019).

- ***Pontos***: tem o intuito de nortear o usuário quanto a sua progressão no jogo, a ideia é que ele se desafie cada vez mais para acumular pontos, é a forma mais clara de observar seus erros e acertos no decorrer das tarefas;

- ***Barra de progresso***: tem o intuito de nortear o jogador quanto seu progresso nos desafios, sem ela os mesmos ficariam perdidos quanto seu posicionamento em determinada partida, e poderiam desistir do jogo faltando poucas tarefas para completar tal desafio.

- ***Tabelas de Classificação***: servem como motivação aos usuários que acumularem pontos, já que, uma tabela é apresentada em cada desafio, contendo a pontuação de todos os participantes de tal sala, e nela os mesmos são ranqueados, proporcionando "*status*" aos usuários mais bem colocados;

- ***Pressão de tempo***: O tempo para realização das tarefas é utilizado para além de nortear o usuário, pontuá-lo de acordo com seu tempo de resposta com pontos extras; a ideia é que o jogador se esforce para responder as questões em tempo hábil e dessa forma consiga obter a quantidade de pontos máximos disponíveis em tal tarefa;

- ***Dicas***: As dicas foram criadas para possibilitar que os usuários as solicitem, "abaixando" indiretamente o nível do desafio sem tirar a satisfação do acerto caso isso aconteça, o intuito é permitir que o usuário se desafie e não se sinta desmotivado pela dificuldade da tarefa, com as dicas pretende-se minimizar esse tipo de situação, vale ressaltar

que elas podem ser solicitadas a qualquer momento do desafio, outro ponto que é relevante mencionar é que caso a solicitação de dicas aconteça uma determinada quantidade de pontos será descontada do valor total do desafio;

- ***Pular desafio***: Os desafios são apresentados aos usuários e neles existe a função de "pular desafio". O propósito de permitir que os usuários pulem desafios é para que eles não se frustrem por não saber resolver alguma tarefa mesmo com as dicas, com isso ele não precisa ficar "parado" em um ponto sem saber como avançar, e embora ele não acumule pontos em determinado desafio, ele conseguirá prosseguir podendo recuperar seus pontos no decorrer das próximas tarefas;

- ***Cooperação***: O modo equipe foi implementado pensando nos usuários que se desenvolvem melhor em equipe, e uma forma de socialização, com essa funcionalidade é possível que os usuários visualizem o mesmo desafio e evoluam juntos, vale ressaltar que a ação de um implica na equipe, os integrantes da equipe podem se comunicar e trocar ideias sobre os desafios propostos antes de chegar a uma resposta através do *chat*;

- ***Socialização***: O chat é a funcionalidade que permite que jogadores se comuniquem e é considerada também uma forma de socialização; nos módulos QUIZ e IDENTIFICAÇÃO com integrantes de sua equipe, enquanto no módulo BATALHA com seu adversário. Através dessa funcionalidade os jogadores podem ficar mais próximos dos demais usuários, tornando o ambiente mais informal;

- ***Acúmulo de Pontos***: O acúmulo de pontos vai além de quantificar os acertos dos usuários, com eles é possível ranquear os jogadores de determinado desafio, e exibi-los na tabela de classificação;

O objetivo central de um sistema gamificado é gerar emoções ao usuário de forma que o mesmo se mantenha interessado e se desafie a continuar jogando enquanto aprende, para isso, é importante que a forma de pontuar os mesmos não seja desmotivante.

O sistema de pontuação do CleanGame baseia-se no progresso do jogador e de suas ações no decorrer de cada atividade. O cálculo da pontuação varia de um módulo para o outro, já que no módulo do Quiz o participante irá acertar, errar ou pular a questão, enquanto no módulo de Identificação e Batalha além das ações citadas ele poderá pedir dicas. As pontuações ocorrem através das seguintes situações:

- **Valor fixo por questão**[3]: Valor inteiro que cada questão vale;
- **Tempo de resposta**[4]: esse item não desconta pontos, é uma pontuação extra. O usuário ganha os pontos caso ele não responda de imediato a questão exibida (o que pode significar que o mesmo chutou a resposta) bem como não gaste um tempo exorbitante para resolver a mesma;
- **Acertos consecutivos**[5]: esse também é um item de premiação extra, que a cada resposta correta consecutiva é acrescido um valor progressivo, caso o usuário erre, essa variável é zerada e é iniciada novamente no próximo acerto consecutivo;
- **Pulo**: No pulo o candidato não perde ponto, apenas deixa de pontuar na questão;
- **Erro**: Caso o usuário responda errado é descontado da pontuação dele um determinado valor, correspondente à 50% do valor fixo por questão;

Os itens mencionados acima são válidos para todos os módulos, a pontuação do módulo de Identificação e Batalha são feitas considerando também:

- **Dicas solicitadas**: são oferecidos três tipos de dicas, das quais cada uma representa um decréscimo na pontuação máxima de determinada questão. As dicas oferecidas são:
  - *Dica de Métricas*: são exibidas as métricas utilizadas para encontrar determinado *smell*. Esta é a dica que menos desconta pontos, representando apenas 5% do valor total da questão;
  - *Dica de Refatoração Sugerida*: é exibida uma possível refatoração para o *smell* existente na classe, o desconto de pontos no valor total das questão equivale à 15%.
  - *Dica de Definição*: é exibida a definição do *smell* da classe, é a dica que mais desconta pontos, representando 30% do valor da questão;

  As dicas são solicitadas de forma sequencial, ou seja, não é possível ao usuário solicitar a dica de definição antes das demais, nem a de refatoração sugerida antes da de métricas. Outro ponto importante no funcionamento da solicitação de dicas, é que se a dica de *Definição* for solicitada em uma rodada, na rodada seguinte as mesmas serão bloqueadas e liberadas posteriormente.

[3] No CleanGame o valor fixo de uma questão foi de 100 pontos.
[4] Varia entre os módulos, já que no módulo QUIZ o tempo de resposta deve ser menor.
[5] O participante ganha um ponto extra a cada acerto consecutivo

Todos os pontos elencados acima servem para nortear a dinâmica esperada dos jogadoress. Ou seja, espera-se que sintam-se motivados a solicitar menos dicas, para que não sejam penalizados e possam ocupar melhores posições no *ranking*; e também que evitem pular desafios ou demorar muito para responder alguma atividade já que, o pulo o deixa com pontuação nula para determinada questão e gerenciar seu tempo de resposta permite que o mesmo receba pontos extras, influenciando, também, em sua posição no *ranking*.

Dessa forma, é almejado que sentimentos como satisfação, competitividade, motivação, socialização e diversão sejam reconhecidos, caracterizando a estética do sistema, já que, conjectura-se que o jogador se sentirá desafiado e motivado a ocupar melhores posições no *ranking*, com isso manterá seu foco na atividade, podendo melhorar o processo de aprendizagem e, ocasionalmente, sentir satisfação com o resultado; as funcionalidades que permitem a comunicação entre jogadores de uma sala, colaboram com a interação social, podendo contribuir, para jogadores que se sintam melhores jogando em equipe e/ou em um duelo direto com outros participantes.

Com esses aspectos definidos a visão concreta do sistema será apresentada na próxima seção (Seção 3.3).

3.3 Visão Geral Concreta da Ferramenta

A visão arquitetural de um sistema é abstrata, apresentando detalhes de implementação, algoritmo e representação de dados, concentrando-se no comportamento e interação de elementos. Sendo considerada como primeiro passo para projetar um sistema (BASS; CLEMENTS; KAZMAN, 2003).

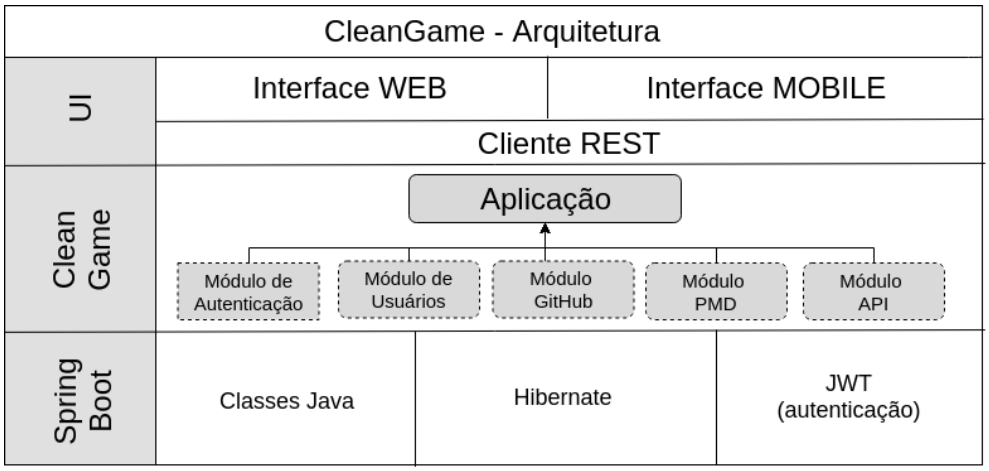
O sistema modelado pela arquitetura de software é geralmente grande e complexo por natureza (TU; GODFREY, 2001). Kruchten (1995). Para remediar esse problema propôs um modelo "4+1"que divide um conjunto específico de preocupação, em diferentes partes:

- visão lógica: descreve o modelo do objeto de *design*, uma visualização lógica; como por exemplo, um diagrama de relacionamento de entidade.
- visão de processo: apresenta os aspectos de simultaneidade e sincronização do sistema.
- visão física: descreve o mapeamento do software no hardware, também reflete os aspectos distribuídos de alguns sistemas.

- visão de desenvolvimento: descreve a organização estática do software em seu ambiente de desenvolvimento.
- cenários e casos de uso: constituem uma visão geral de como todas as quatro visualizações funcionam juntas.

A Figura 3.4 apresenta a organização estática do CleanGame, que, segundo Kruchten (1995), é descrita como "visão de desenvolvimento".

Figura 3.4 – Arquitetura do Sistema



Fonte: Do autor (2019).

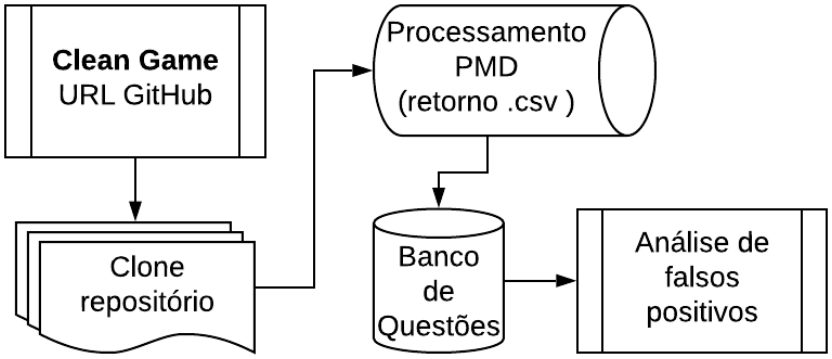
Como pode ser observado, o sistema foi projetado para possuir um *Back-End* independente, desenvolvido em linguagem *Java* utilizando o *framework Spring Boot* e operando sobre padrão REST[6], possibilita diferentes implementações de *Front-End*, construído com o *framework javascript AngularJs*, sem necessidade de alterações no *Back-End*. As tecnologias escolhidas para desenvolvimento do sistema foram determinadas baseando, principalmente, no conhecimento prévio da autora sobre as mesmas.

É possível visualizar na Figura 3.4 que existem os módulos GitHub e PMD em sua implementação. A criação desses módulos foi necessária para que sistema conversasse com essas ferramentas externas na geração de conteúdo.

A geração de conteúdo de forma automática acontece no módulo de identificação do CleanGame que, como mencionado, é totalmente integrado à API do GitHub[7] e ao plugin PMD[8]. Assim sendo, durante a criação de uma sala no módulo de identificação, o usuário precisa fornecer a URL de um repositório Java do GitHub.

[6] (*Representational State Transfer*)
[7] Interface de programação de aplicativos fornecida pelo GitHub
[8] http://pmd.sourceforge.net/

Figura 3.5 – Integração GitHub e PMD



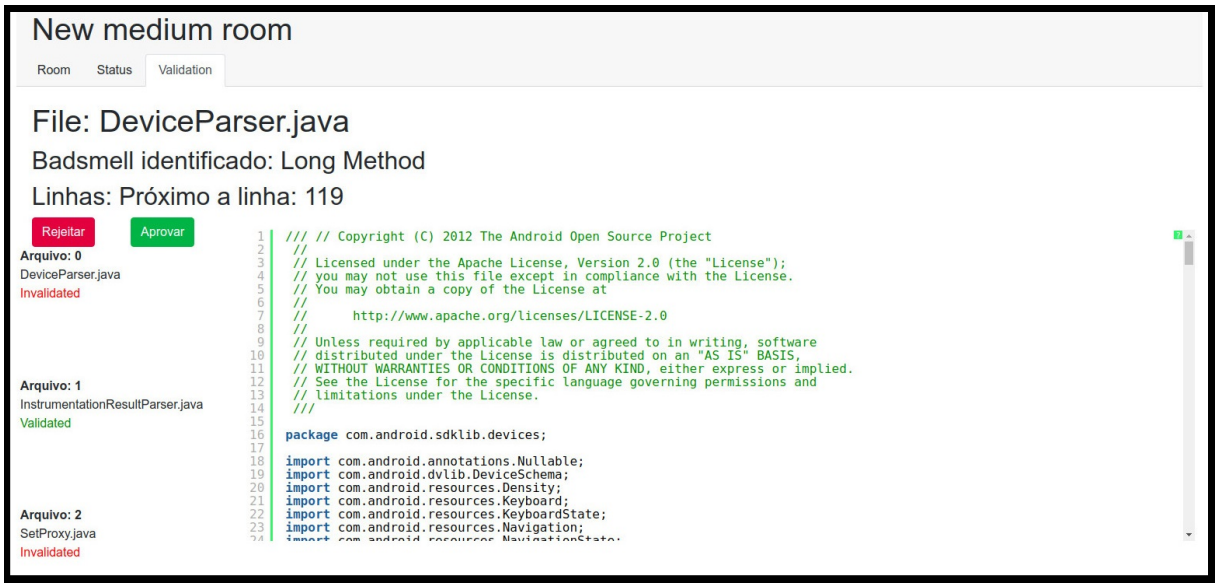
Fonte: Do autor (2019).

Como pode ser observado na Figura 3.5, essa integração, chamada oráculo, é responsável por clonar o repositório fornecido através da URL, gerar as questões e dicas que serão fornecidas ao usuário e validar suas respostas.

Após a clonagem do repositório o sistema faz uma leitura através da ferramenta PMD, para que sejam identificados os *code smells* existentes em tal projeto; um arquivo é retornado ao sistema mencionando a linha do *code smell* sua natureza e nível de importância, com essas informações o banco de questões com as dicas é gerado.

Para excluir os falsos positivos que podem ser fornecidos pela PMD, o sistema solicita que o administrador da sala valide as informações fornecidas pelo sistema como pode ser observado na Figura 3.6.

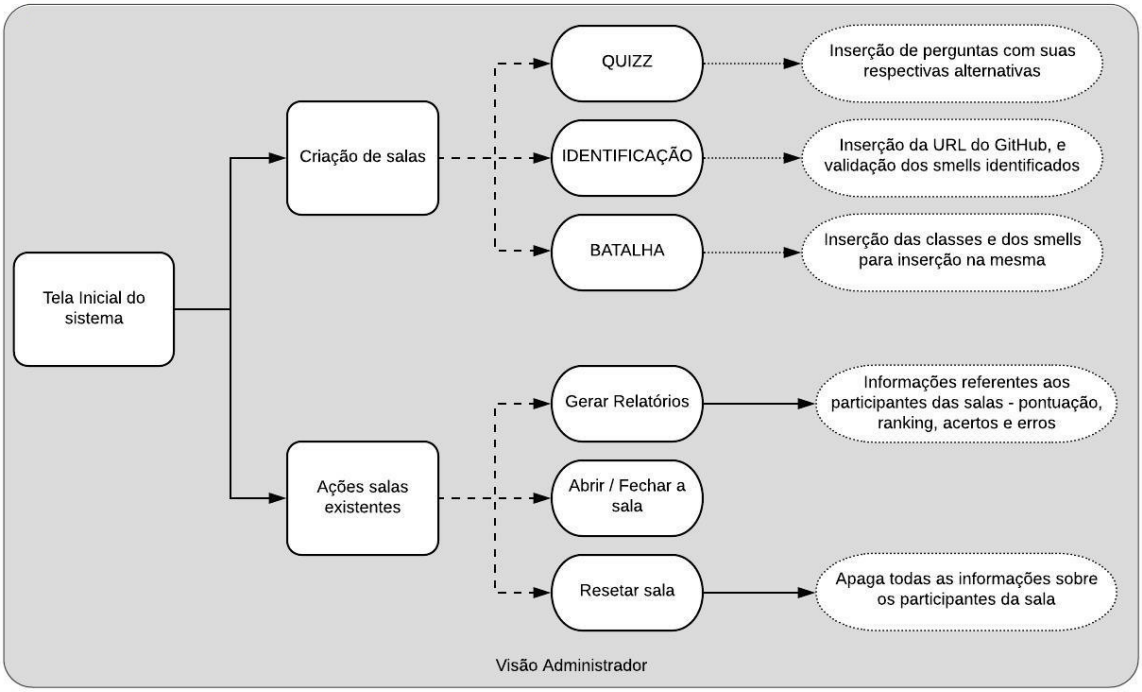
Figura 3.6 – Módulo Identificação



Fonte: Do autor (2019).

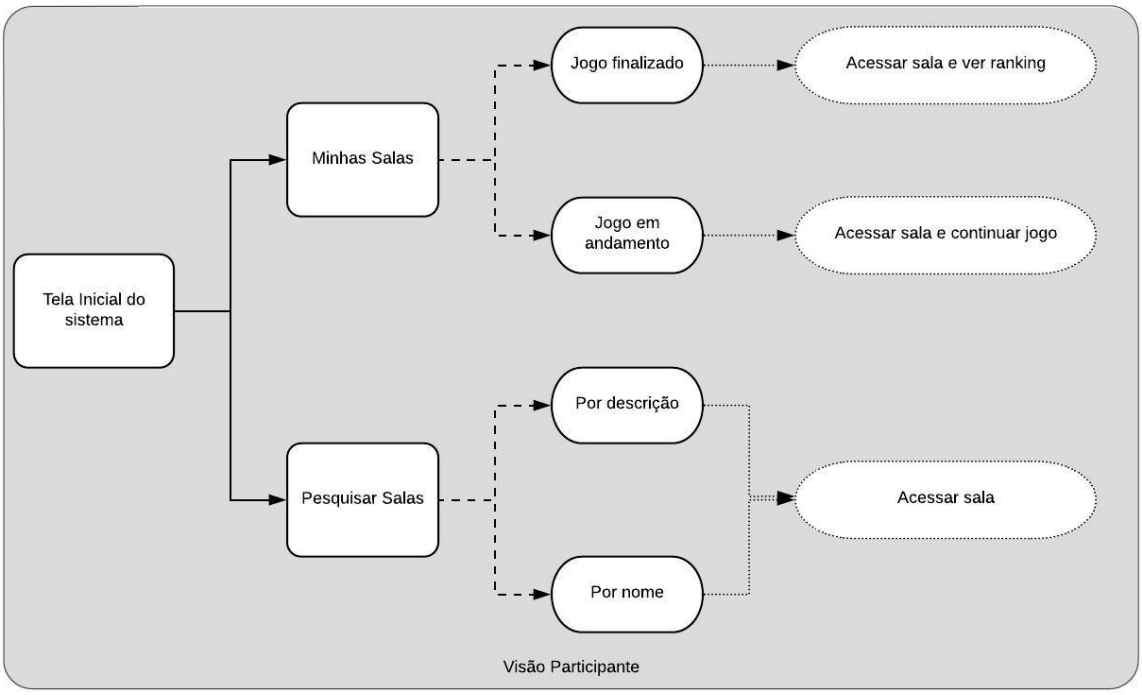
O Administrador é um dos dois perfis existentes do sistema, como pode ser observado nas Figuras 3.7 e 3.8 existe um fluxo de ações para cada tipo de usuário, embora um administrador seja, também, um participante do sistema. Abaixo será detalhada a existência desses perfis.

Figura 3.7 – Fluxograma Administrador



Fonte: Do autor (2019).

Figura 3.8 – Fluxograma Participante



Fonte: Do autor (2019).

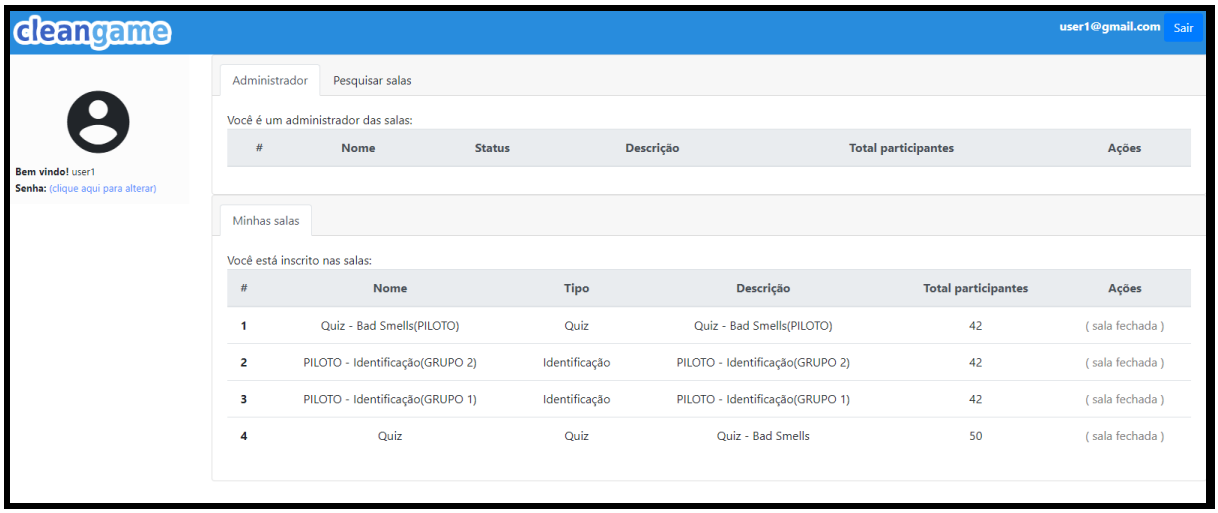
Perfis de Usuários

Os perfis de usuários foram criados almejando que a interação dos mesmos com o sistema fosse além de participar dos desafios, e esses pudessem também usá-lo para criar seus próprios desafios de acordo com os módulos disponíveis.

No sistema um perfil é dito *ADMINISTRADOR*, quando um usuário com perfil *PARTICIPANTE* cadastra uma sala em qualquer dos módulos, ou seja, um perfil administrador, sempre será um participante. Essa funcionalidade proporciona que usuários consigam criar seus próprios desafios, além de auxiliar no abastecimento do banco de questões do sistema.

- **Participante:** No momento que um usuário se cadastra no sistema se torna apto a participar de qualquer sala, é disponibilizado a ele, mesmo antes de acessar a sala, informações referentes a natureza (módulo), descrição (conteúdo contido), status (se a mesma está acessível no momento ou não) e número de participantes, como pode ser observado na Figura 3.9.

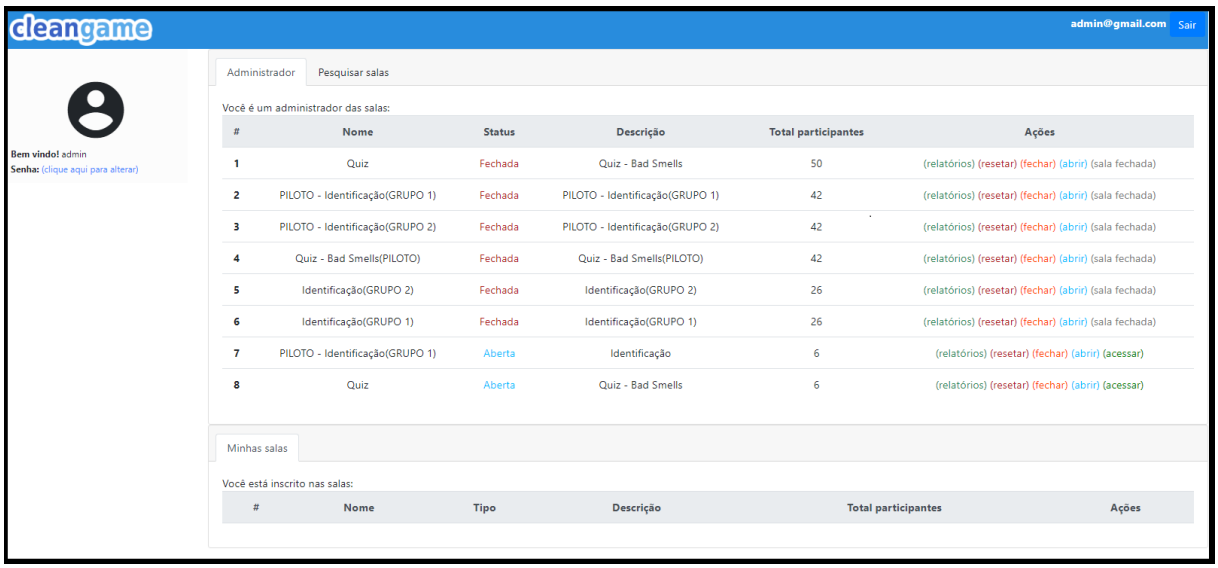
Figura 3.9 – Visão participante



Fonte: Do autor (2019).

- **Administrador:** Quando um participante cadastra uma sala, ele se torna administrador da mesma, as quais são listadas à ele com suas informações básicas (nome, status,descrição, total de participantes), além de algumas ações como: gerar relatórios, resetar, fechar ou abrir sala. A forma de inserção de salas varia de módulo para módulo, o funcionamento dessa ação será detalhada na Seções 3.4. A visão das salas do administrador é apresentada na Figura 3.10.

Figura 3.10 – Visão administrador



Fonte: Do autor (2019).

3.4 Módulos do Sistema

Como já mencionado, o CleanGame é compostos por três módulos independentes (QUIZ, IDENTIFICAÇÃO e BATALHA) com salas previamente cadastradas, em ambos os módulos. Os participantes podem interagir em duas ou mais salas de uma vez, não necessitando finalizar um desafio para iniciar outro. Caso o participante pause seu desafio em determinada sala, o jogo retornará ao ponto em que foi parado.

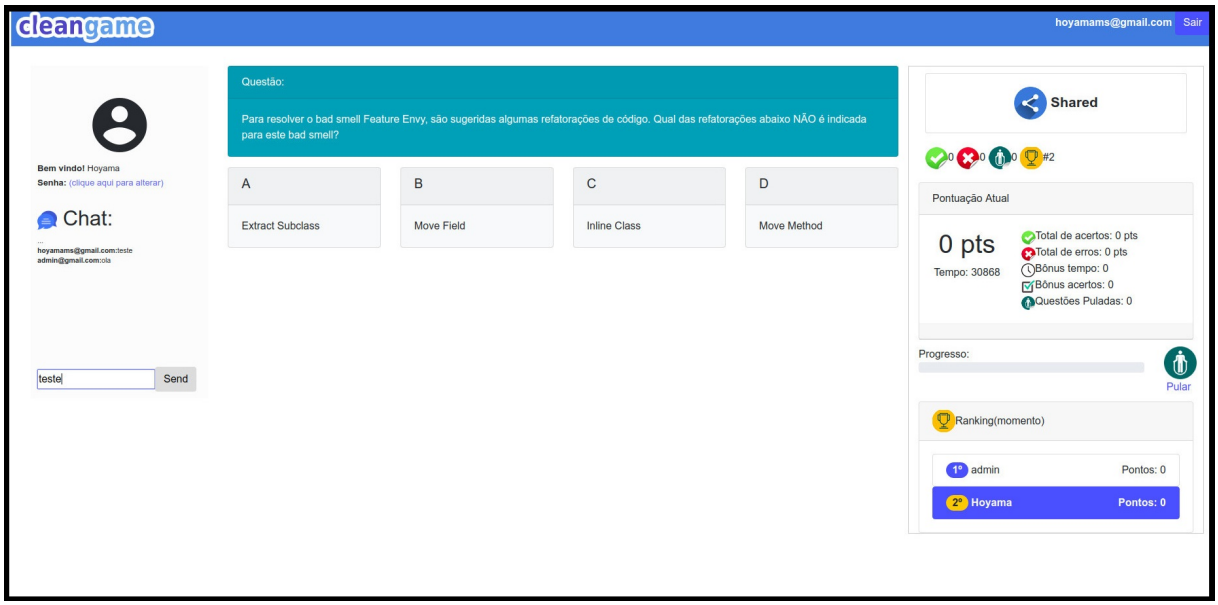
Os módulos *Quiz* e *Identificação* apresentam a funcionalidade de *shared*, ou seja, é possível que o desafio de uma sala seja compartilhado para dois ou mais jogadores, e esses realizem os desafios de forma conjunta. A partir do momento que uma sala é compartilhada é disponibilizado aos jogadores um *chat*, onde eles podem discutir sobre as questões apresentadas para acordarem sobre suas respostas. O módulo *Batalha* conta somente com a funcionalidade de *chat* para proporcionar interação entre os oponentes.

No decorrer dos desafios é permitido que os jogadores solicitem dicas, porém caso isso aconteça a pontuação do mesmo será afetada independentemente do êxito em tal questão. Outra ação que o jogador pode realizar no decorrer dos desafios é pular uma questão, esse no entanto não desconta pontos do usuário, porém se o jogador pulou uma questão não é permitido que faça a mesma ação na questão subsequente.

3.4.1 Módulo QUIZ

Nesse módulo são apresentadas perguntas com suas possíveis respostas, como pode ser observado na Figura 3.11. É o módulo mais simples do sistema.

Figura 3.11 – Módulo Quiz

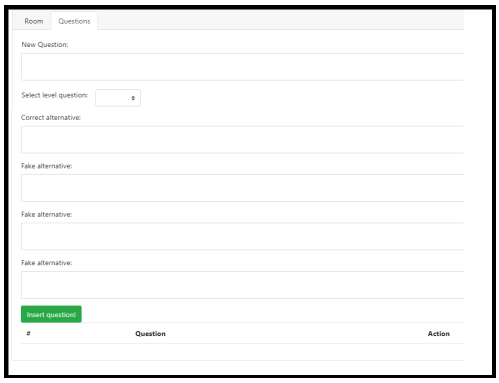


Fonte: Do autor (2019).

Inserção de Conteúdo

No *QUIZ*, é solicitado que o administrador forneça o nome da sala, sua descrição, a pergunta com suas alternativas falsas (três), resposta correta e o nível da questão (fácil, moderada, difícil). A quantidade de questões cadastradas em uma sala fica a critério do administrador.

Figura 3.12 – Módulo Quiz - Inserção



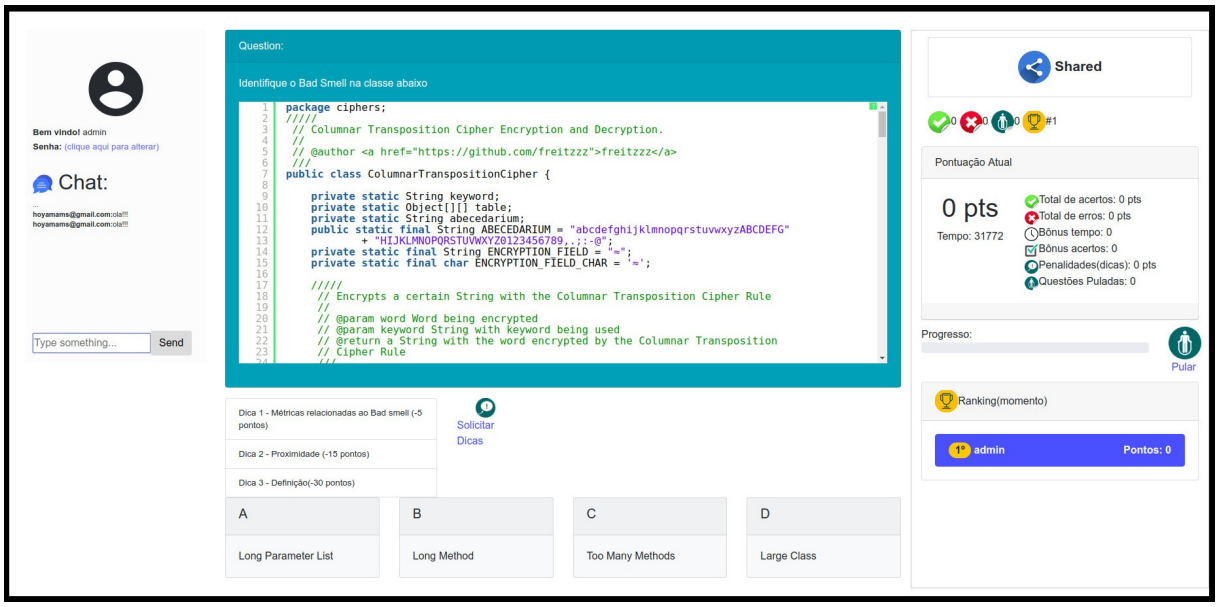
Fonte: Do autor (2019).

3.4.2 Módulo IDENTIFICAÇÃO

Nesse módulo, é apresentada uma classe Java para que o jogador selecione qual é o *code smell* existente na mesma. São disponibilizadas, caso necessite, dicas quanto às métricas, proximidade ou definição do *code smell* existente.

Inserção de Conteúdo

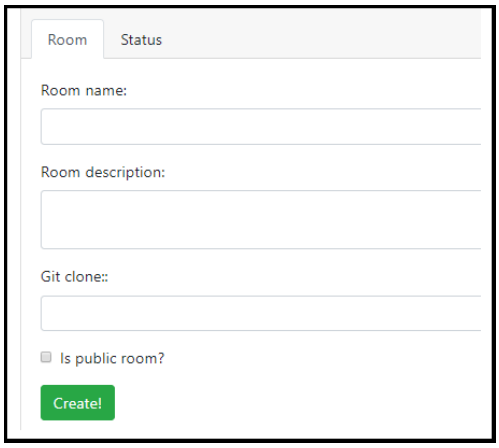
Figura 3.13 – Módulo Identificação



Fonte: Do autor (2019).

Na *IDENTIFICAÇÃO*, é solicitado que o administrador forneça o nome da sala, a descrição e uma URL do GitHub - Figura 3.14 (todo o conteúdo é gerado pelo sistema a partir da URL fornecida - como apresentado na Seção 3.3). Após a geração do conteúdo é solicitado que o administrador valide as classes com *code smells* identificados pelo PMD, para que os falsos positivos sejam eliminados, como é apresentado na Figura 3.6. As classes são disponibilizadas apenas quando já estiverem validadas, ou seja, a quantidade de questões nesse módulo também fica a critério do administrador. Note que, os *smells* são identificados de forma totalmente automática pelo CleanGame com a ajuda da ferramenta PMD. Além disso, o administrador pode escolher quais *smells* identificar no momento em que os valida.

Figura 3.14 – Módulo Identificação - Inserção

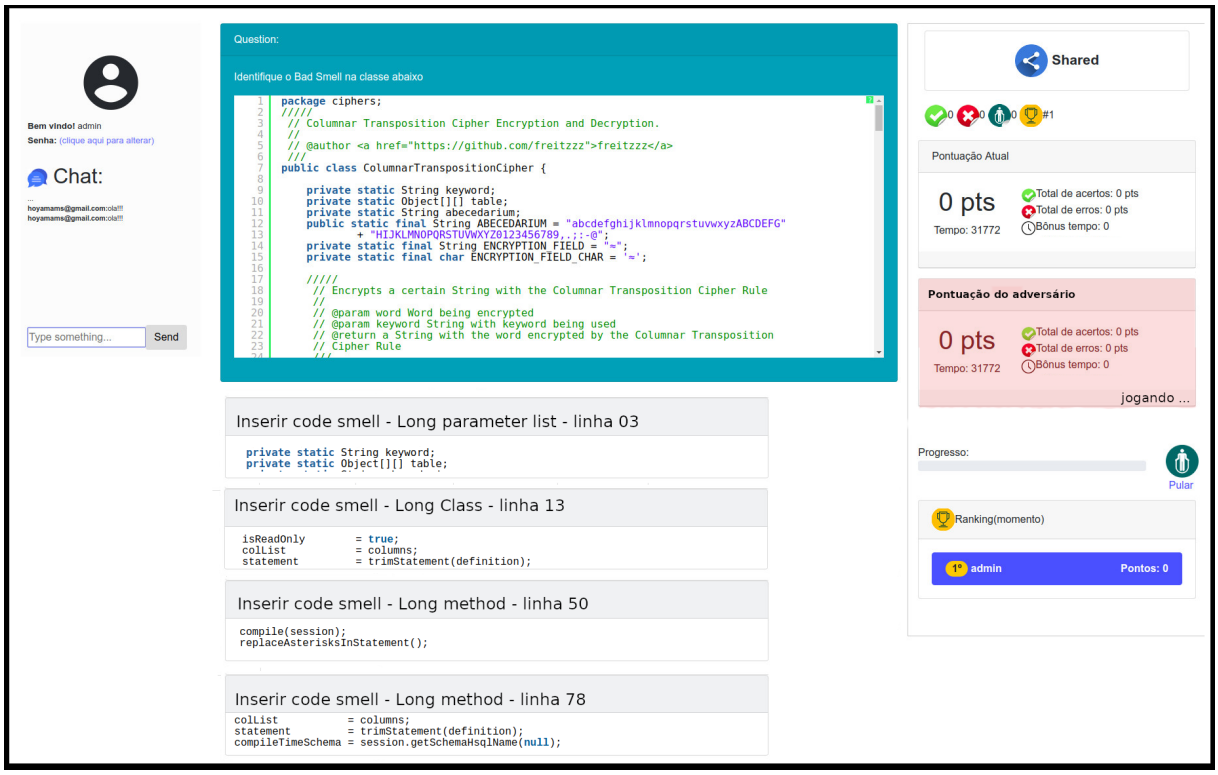


Fonte: Do autor (2019).

3.4.3 Módulo BATALHA

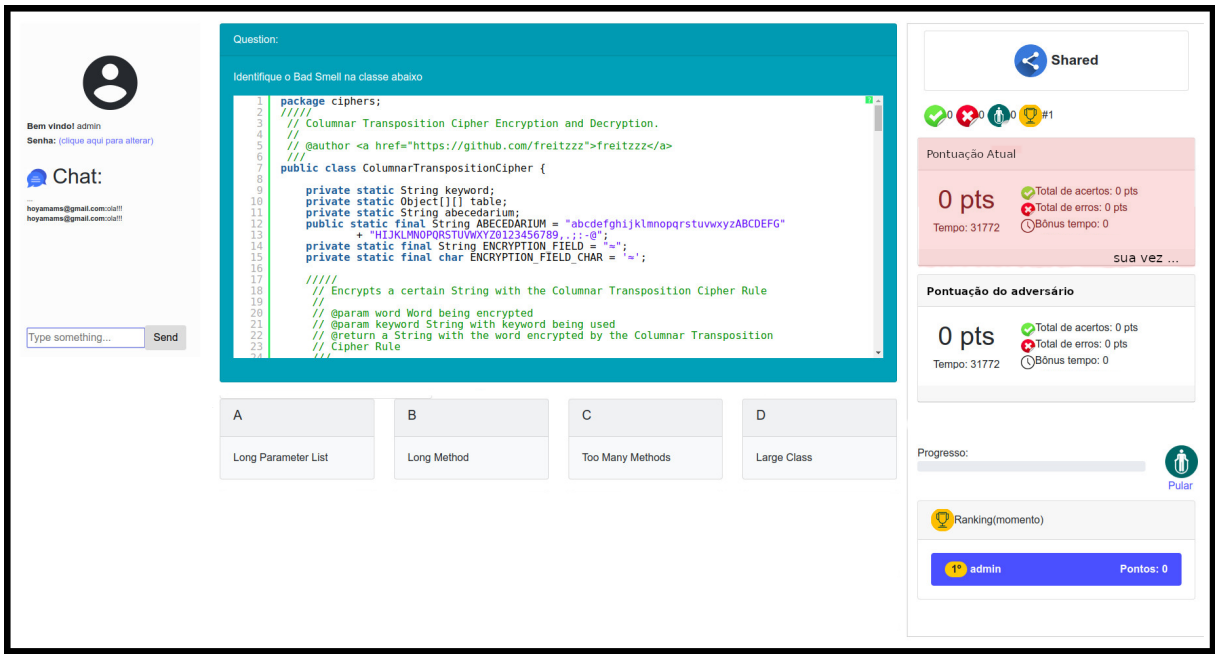
Nesse módulo é apresentada uma classe para cada um dos envolvidos na batalha e dispostos os trechos de códigos que poderão ser acrescidos ao código original, como pode ser observado na Figura 3.15, a cada rodada os jogadores deverão inserir o código previamente cadastrado, e em seguida identificar o *code smell* adicionado por seu oponente, como apresentado na Figura 3.16. Assim que o *code smell* é adicionado por ambos os jogadores no código, ocorre a troca de códigos entre eles para que identifiquem o *smell* adicionado pelo oponente.

Figura 3.15 – Módulo Batalha - Ataque



Fonte: Do autor (2019).

Figura 3.16 – Módulo Batalha - Defesa

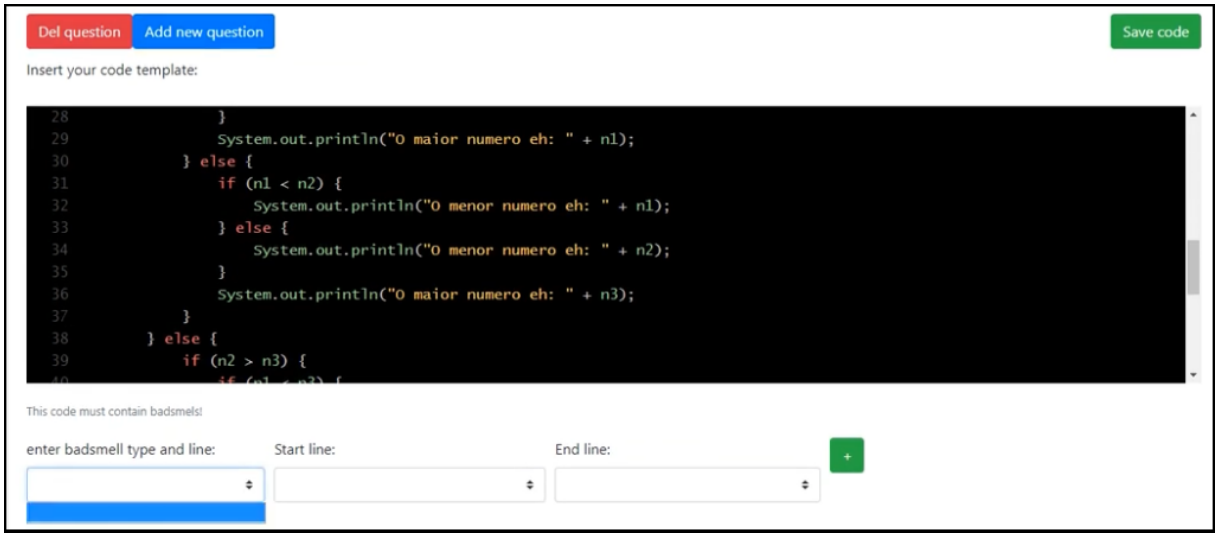


Fonte: Do autor (2019).

Inserção de Conteúdo

Na *BATALHA* a inserção de conteúdo é parecida com o módulo Quiz, nele o administrador deve fornecer as classes que serão exibidas aos jogadores com os possíveis *code smells* a serem inseridos, a linha de inserção e o tipo do mesmo, como apresentado na Figura 3.17.

Figura 3.17 – Módulo Batalha - Inserção



Fonte: Do autor (2019).

3.5 Considerações Finais

Este capítulo apresentou o CleanGame, um sistema gamificado para aprendizado e fixação dos conceitos sobre *code smells*, desde a formulação abstrata ao produto final. Com os conceitos aqui expostos, além da utilização do mesmo para o propósito do trabalho, é possível que educadores adaptem sistemas e utilizem as diretrizes para gamificar sistemas de diferentes seguimentos em Engenharia de Software.

No capítulo seguinte, é apresentada a avaliação do mesmo na perspectiva dos discentes com relação ao seu uso como ferramenta complementar de ensino.

4 AVALIAÇÃO DO SISTEMA PROPOSTO - CLEANGAME

Este capítulo descreve o planejamento, execução e análise do experimento realizado para validar a eficácia do uso do CleanGame como ferramenta de suporte para o aprendizado e fixação na identificação de *code smells*. A avaliação se deu através de um experimento realizado com estudantes do curso de computação de uma universidade federal, e foi elaborado em três partes, em que os alunos executavam uma atividade de maneira tradicional e uma com a ferramenta gamificada; por fim os alunos responderam a um questionário sobre as abordagens trabalhadas no experimento.

Este capítulo está organizado da seguinte forma: A Seção 4.1 descreve a configuração do experimento com as questões de pesquisa. A Seção 4.2 descreve o escopo do estudo com a formulação das hipóteses. A Seção 4.3 apresenta a seleção dos sujeitos, seguida da descrição do *design* do experimento na Seção 4.4. Na Seção 4.5 é apresentado o planejamento e execução do experimento. A Seção 4.6 relata as ameaças à validade do experimento. Com isso, os resultados do experimento são apresentados na Seção 4.7, seguidos da configuração e resultados da pesquisa atitudinal na Seção 4.8. Por fim, a Seção 4.9 fecha o capítulo com as considerações finais.

4.1 Configuração do Experimento

A gamificação é considerada adequada para envolver usuários em determinadas tarefas, como mencionado anteriormente. Com base nessa premissa, é conjecturado que a mesma seja pertinente para envolver alunos com tópicos relacionados à refatoração, como a identificação de *code smells*, especialmente quando usada como ferramenta de apoio. Desse modo, para termos de validação, é necessário explorar se a gamificação pode ter um impacto positivo no reforço pós-treinamento em comparação com uma abordagem tradicional.

O pós-treinamento tradicional para avaliar o desenvolvimento dos alunos em atividades de identificação de *code smells* envolve tarefas práticas que necessitam da leitura do código-fonte para identificar tais *smells*. Geralmente, essas tarefas são realizadas em ambientes de desenvolvimento integrado (IDE), que permite a navegação mais fácil pelos códigos (SPINELLIS, 2012).

A falta de diretrizes e elementos para manter os alunos envolvidos nas atividades, tornam o pós-treinamento tradicional de identificação de *code smells* cansativo. Diante disso, é admitido que uma abordagem gamificada pode ser empregada para mitigar esses problemas.

Para investigar os benefícios proporcionados por um ambiente gamificado, uma ferramenta que suporta atividades pós-treinamento centradas na identificação de *code smells* foi desenvolvida. No contexto da ferramenta, as atividades pós-treinamento seguem uma abordagem de *design* de jogo, ou seja, elas utilizam elementos de gamificação, como tabela de classificação e recompensas.

Dessa forma, um experimento foi projetado e uma pesquisa de atitude foi realizada para responder às seguintes questões de pesquisa (**RQs**):

RQ_1: *A gamificação tem um impacto positivo em como os alunos identificam code smells durante as atividades pós-treinamento?*

É admitido que os alunos serão mais bem sucedidos com uma abordagem baseada em jogos para a identificação de *smells*, pela existência de evidências que elementos de gamificação, como pontos e tabelas de classificação, transmitem um senso de competência aos mesmos e aumentam a motivação intrínseca, melhorando assim o desempenho.

De acordo com essas evidências, é presumido que um ambiente gamificado é mais eficaz para transmitir habilidades de identificação de *code smells* e manter os alunos envolvidos do que uma abordagem mais tradicional para a identificação dos mesmos (ou seja, orientada por IDE).

Portanto, $\mathbf{RQ}_1$ se resume a examinar o impacto e a solidez da gamificação ao envolver os alunos na identificação dos *code smells*. No contexto do experimento, é usada a taxa média de respostas corretas (ou seja, quantidade de *code smells* identificados corretamente) para medir a eficácia da gamificação.

$\mathbf{RQ}_2$: *Os alunos têm uma atitude positiva em relação a uma experiência de aprendizado baseada em jogos?*

A eficácia de uma abordagem pós-treinamento é fortemente influenciada pelas atitudes adotadas em relação à forma como o conteúdo é apresentado. Assim, $\mathbf{RQ}_2$ investiga a perspectiva dos sujeitos sobre a gamificação como uma abordagem pós-treinamento para a identificação de *code smells*.

$\mathbf{RQ}_3$: *Quais são as vantagens e desvantagens de uma abordagem pós-treinamento gamificada?*

Pretende-se avaliar, também, os prós e os contras da gamificação como uma abordagem de reforço pós-treinamento do ponto de vista dos alunos. Para responder $\mathbf{RQ}_2$ e $\mathbf{RQ}_3$, uma

pesquisa de atitude é realizada para avaliar a opinião, o nível de satisfação e a atitude geral dos alunos em relação à abordagem pós-treinamento.

4.2 Escopo

A organização utilizada para definir os objetivos do experimento foi a proposta pelo Objetivo / Pergunta / Métrica (GQM) (WOHLIN et al., 2012). Seguindo esse modelo de definição de objetivo, o escopo do estudo pode ser resumido conforme descrito abaixo.

e **Analisar** a abordagem gamificada
para fins de avaliação
com relação à eficácia pós-treinamento
do ponto de vista do pesquisador
no contexto de estudantes que procuram *code smells*.

Como mencionado, no contexto do experimento, é avaliado o impacto da abordagem em termos de sua eficácia como acompanhamento de treinamento. É usada uma abordagem baseada em IDE para comparação de referência.

4.2.1 Formulação de Hipóteses

A previsão para $\mathbf{RQ}_1$ é enquadrada em: a abordagem pós-treinamento gamificada é mais eficaz do que uma abordagem orientada a IDE. Como mencionado, para responder a $\mathbf{RQ}_1$, é avaliado a eficácia das abordagens pós-treinamento em termos de quantidade de *code smells* identificados corretamente (CICS). Então, $\mathbf{RQ}_1$ foi transformada nas seguintes hipóteses:

Hipótese nula, H $_{0-CICS}$: não há diferença entre uma abordagem gamificada e uma abordagem baseada em IDE para identificação de *code smells* em termos da quantidade de *smells* identificados corretamente pelos alunos durante o pós-treinamento.

Hipótese alternativa, H $_{1-CICS}$: os alunos são capazes de identificar mais *code smells* em um ambiente gamificado do que através de uma abordagem tradicional (ou seja, orientada por IDE).

Seja μ a quantidade média de *code smells* identificados corretamente. Portanto, $\mu_{CleanGame}$ e μ_{IDE} denotam a quantidade média de *code smells* identificados corretamente pelos alunos que

usam o CleanGame e a quantidade média de *smells* identificados usando uma abordagem baseada em IDE. Então, o conjunto de hipóteses acima mencionado pode ser formalmente declarado como:

$$\mathbf{H}_{0-CICS}\colon\ \mu_{CleanGame} = \mu_{IDE}$$

e

$$\mathbf{H}_{1-CICS}\colon \mu_{CleanGame} > \mu_{IDE}$$

4.3 Seleção de Sujeitos

O experimento foi realizado na Universidade Federal de Minas Gerais com estudantes de Ciência da Computação, mais especificamente, estudantes de graduação, mestrado e doutorado. Vale ressaltar que este estudo pode ser classificado como um quase experimento, devido à falta de randomização dos participantes. A capacidade de generalizar a partir desse contexto específico é elaborada na Seção 4.6.

Todos os sujeitos já tinham experiência anterior com Java e programação orientada a objetos. O conhecimento prévio sobre refatoração e conceitos relacionados à ela (*code smells*) não era obrigatório. Foi solicitado que todos os participantes assinassem um termo de consentimento antes de participar do experimento.

4.4 Design do Experimento

O experimento possui duas formas de abordagem: uma é a abordagem de reforço pós-treinamento através da qual os sujeitos tentam identificar os *code smells* através do CleanGame (ou seja, a maneira gamificada de ensinar e apoiar a identificação dos *code smells*) e a abordagem baseada em IDE(atribuição prática usando uma IDE). A experiência dos sujeitos não foi usada como fator de bloqueio, não sendo solicitado aos participantes que preenchessem um questionário pré-experimental, porque deliberou-se não estratificar ainda mais a amostra em grupos com níveis de experiência semelhantes. Portanto, supõe-se que os sujeitos desse experimento tenham experiência e nível equivalentes.

Foi utilizado um delineamento cruzado randomizado para que todos os sujeitos pudessem ser expostos a ambas as abordagens pós-treinamento. Ou seja, todos os participantes foram designados para usar o CleanGame, bem como a abordagem pós-treinamento baseada em IDE. Ambos os grupos passaram pelos mesmos programas Java e *code smells*.

4.5 Instrumentação

Na fase de introdução, foi ministrada ao participantes uma aula sobre refatoração e identificação de *code smells*. Em sequência, foi realizada uma randomização para que os sujeitos pudessem ser expostos a ambas as tarefas pós-treinamento, ou seja, a abordagem gamificada e a tradicional. Mais especificamente, os indivíduos foram divididos aleatoriamente em dois grupos e designados para concluir as tarefas de identificação de *code smells* usando cada abordagem da seguinte maneira: um grupo executou a identificação de *smells* usando uma IDE seguido pela identificação de *code smells* usando CleanGame; o outro grupo realizou a identificação de *code smells* com o CleanGame seguido pela identificação de *smells* usando uma IDE. Portanto, a resposta é medida duas vezes em cada sujeito.

Após a randomização, os sujeitos designados para utilizar a CleanGame em primeiro momento participaram de uma curta sessão de treinamento, na qual foram apresentados a cada recurso da ferramenta. Durante esta sessão de treinamento, os participantes identificaram alguns *code smells* usando o CleanGame. O objetivo era permitir que os mesmos se familiarizassem com a interface gráfica do usuário (GUI) da ferramenta. Além disso, ao longo desta sessão de treinamento, os sujeitos foram autorizados a fazer qualquer pergunta sobre o CleanGame. Nenhuma assistência adicional foi fornecida aos indivíduos designados para realizar tarefas pós-treinamento usando a abordagem tradicional (ou seja, orientada por IDE).

Como foi usado um *design* aleatório, em um estágio posterior, o grupo que participou primeiro das tarefas de identificação de *smells* usando o CleanGame foi designado para executar tarefas de identificação de *code smells* usando a abordagem tradicional. Por sua vez, o grupo selecionado inicialmente para a abordagem tradicional foi introduzido no CleanGame (como para o primeiro grupo, houve uma breve sessão de treinamento) e começou a identificar os *code smells* usando a abordagem gamificada.

As vantagens de aplicar cada abordagem de identificação de *code smells*, conforme percebida pelos sujeitos, foram investigadas por meio de um pós-questionário entregue após a realização do experimento (fase de finalização). Além disso, o mesmo questionário também foi usado para coletar informações adicionais dos participantes sobre os principais obstáculos / inibidores da aplicação de ambas as abordagens para identificação de *code smells*.

4.6 Ameaças à Validade

Como em qualquer estudo empírico, esse experimento apresenta várias ameaças à validade. Nesta seção, são descritas as quatro principais ameaças que podem comprometer o experimento: *(i)* interno, *(ii)* externo, *(iii)* conclusão e *(iv)* construção.

A validade interna tem a ver com a confiança que pode ser colocada na relação causa-efeito entre os tratamentos e as variáveis dependentes no experimento. A validade externa diz respeito à generalização: se a relação causa-efeito entre os tratamentos e as variáveis dependentes pode ser generalizadas fora do escopo do experimento.

A validade da conclusão está centrada nas conclusões que podem ser extraídas da relação entre tratamento e resultado. Finalmente, a validade de construção está próxima da relação entre teoria e observação: se os tratamentos refletem adequadamente a causa e a adequação dos resultados para representação do efeito.

4.6.1 Validade Interna

A questão do viés de seleção foi atenuada através da randomização. No entanto, como foi assumido que todos os sujeitos têm antecedentes semelhantes, nenhum fator de bloqueio foi aplicado para minimizar a ameaça de possíveis variações no desempenho dos sujeitos. Portanto, não pode-se descartar a possibilidade de que alguma variabilidade no desempenho dos mesmos decorra de seus conhecimentos e experiências anteriores.

Outra possível ameaça à validade interna tem a ver com os arquivos que contêm os *code smells* que foram usados no experimento: se fossem usados outros arquivos, os resultados poderiam ter sido diferentes. No entanto, houve a tentativa de diminuir essa ameaça selecionando arquivos com *code smells* representativos do nível de experiência de estudantes de graduação e pós-graduação. Especificamente, foram selecionados *code semlls* da Landfill (PALOMBA et al., 2015b), que é uma plataforma baseada na Web para compartilhar e validar conjuntos de dados de *code smell*. De acordo com o conhecimento do autor a Landfill compreende a maior coleção disponível publicamente de *smells* validados manualmente.

4.6.2 Validade Externa

A principal ameaça à validade externa dos resultados é a amostra: como mencionado, o experimento foi realizado por estudantes, portanto todas as disciplinas têm formação aca-

dêmica. Assim, os *insights* obtidos com o experimento podem ser generalizados apenas para configurações semelhantes (ou seja, no contexto de alunos com experiência semelhante). Houve ciência de que é necessária uma replicação adicional do experimento para estabelecer resultados mais conclusivos: para aumentar a validade externa, é preciso replicar o estudo com uma amostra maior.

Além disso, a amostra consistiu apenas de estudantes. Portanto, não pode-se afirmar se o CleanGame também seria capaz de envolver os profissionais, ajudando-os a aprimorar suas habilidades na identificação de *code smells*. Diante disso, não é possível descartar a ameaça de que os resultados poderiam ter sido diferentes se os profissionais fossem selecionados como sujeitos.

Portanto, pode valer a pena replicar o experimento com uma amostra mais diversa (incluindo profissionais) para corroborar as descobertas. Também vale ressaltar que os sujeitos da amostra podem ter maior afinidade com os videogames e, portanto, melhores atitudes em relação a uma abordagem baseada na gamificação do que a população em geral.

Além disso, com relação à generalização das descobertas, sabe-se que o escopo do experimento foi limitado apenas a programas Java. Para estudos futuros, pretende-se replicar o experimento usando programas escritos em outras linguagens de programação. Também é importante notar que os *code smells* foram concentrados nos encontrados em programas de código aberto, portanto, não pode-se especular sobre como os resultados do experimento seriam diferentes ao levar em conta o software em escala industrial. No entanto, foi conjecturado que o uso de programas que são muito complexos pode prejudicar o aprendizado.

4.6.3 Validade de Conclusão e de Construção

A abordagem utilizada para analisar os resultados do experimento representa a principal ameaça às conclusões do estudo: os resultados são discutidos apresentando estatísticas descritivas e testes estatísticos de hipóteses.

Relacionado a validade de construção, existe a possibilidade de que as medidas empregadas no experimento possam não ser apropriadas para quantificar os efeitos propostos a investigar. Por exemplo, a quantidade de respostas corretas pode não ser o único nem o mais importante preditor da eficácia e engajamento pós-treinamento. Se as medidas usadas não refletem o construto teórico alvo, os resultados do experimento podem ser menos confiáveis.

Uma maneira de estender o estudo é examinar quais outras medidas podem ser relevantes para um modelo de eficácia pós-treinamento no contexto da identificação de *code smells*. Além disso, como o experimento ocorreu como parte de uma disciplina, em que os alunos recebem notas, suas respostas podem ser influenciadas na esperança de obter uma nota melhor. Para atenuar essa ameaça, os participantes tiveram certeza de que o experimento não teria nenhum efeito em suas notas.

4.7 Resultados do Experimento

Nesta seção, são apresentados os resultados do experimento realizado. Primeiro, foram delineadas algumas estatísticas descritivas, depois apresentado o teste de hipóteses.

4.7.1 Estatística Descritiva

Como mencionado, foi empregado um delineamento cruzado aleatório, de modo que os sujeitos de ambos os grupos fossem expostos às duas abordagens (Seção 4.5). A Tabela 4.1 apresenta resultados detalhados do desempenho dos dezoito(18) participantes ao usar o CleanGame para identificar *code smells*. Na Tabela 4.2, é apresentado o resumo de como os sujeitos se saíram ao identificar os *code smells* via IDE.

Conforme mostrado nas Tabelas 4.1 e 4.2, em média, os sujeitos foram capazes de identificar aproximadamente o dobro de *code smells* usando o CleanGame (4.94) em comparação com a IDE (2.39). Além disso, o participante com melhor desempenho conseguiu identificar corretamente 8 *code smells*, em 10, usando o CleanGame.

Conforme apresentado na Figura 4.1, os participantes do grupo 1 tiveram um desempenho um pouco melhor na identificação de *code smells* através do CleanGame, já com os participantes do grupo 2 o desempenho foi significativamente melhor através do CleanGame.

Tabela 4.1 – Desempenho da identificação de *smells* dos dois grupos experimentais usando o CleanGame.

Grupo 1							
Participante	**Respostas Corretas**	**Respostas Incorretas**	**Pulos**	**Dica - Métrica**	**Dica - Refatoração**	**Dica - Definição**	**Tempo Médio†**
#1	6	4	0	8	2	0	476
#2	6	4	0	6	4	4	2,593
#3	5	5	0	6	3	0	1,336
#4	5	5	0	9	4	0	640
#5	5	5	0	6	3	3	674
#6	4	6	0	4	4	3	1,601
#7	2	7	1	1	1	0	1,198
#8	1	9	0	3	1	0	1,231
#9	3	7	0	3	2	1	891
Grupo 2							
Participante	**Respostas Corretas**	**Respostas Incorretas**	**Pulos**	**Dica - Métrica**	**Dica - Refatoração**	**Dica - Definição**	**Tempo Médio†**
#1	8	2	0	8	4	0	897
#2	7	3	0	1	1	0	1,259
#3	7	3	0	1	0	0	2,349
#4	6	4	0	8	4	0	1,317
#5	6	4	0	9	4	1	993
#6	5	5	0	9	2	0	1,411
#7	5	5	0	8	2	1	1,951
#8	4	6	0	0	0	0	960
#9	4	6	0	1	1	1	943
Estatística descritiva para ambos os grupos experimentais							
Min	1	2	0	0	0	0	476
Max	8	9	1	9	4	4	2,593
Média	4.94	5.00	0.05	5.06	2.33	0.78	1,262.22
Dev.Padrão	1.76	1.68	0.24	3.30	1.46	1.27	566.10
† O tempo médio é indicado em segundos.							

Fonte: Santos et al. (2019).

Ao combinar o desempenho de ambos os grupos com os dois tratamentos experimentais, os resultados parecem indicar que o CleanGame permitiu que os participantes fossem mais eficazes na identificação de *code smells* (Figura 4.1).

Os sujeitos estavam mais aptos a pular as tarefas de identificação de *smells* ao usar um IDE. Usando a IDE, os sujeitos ignoraram, em média, 1,22 tarefas: o sujeito #5 do grupo 2 teve o maior número de perguntas respondidas nesse grupo, 6 perguntas foram ignoradas. Os participantes que tiveram um número maior de perguntas ignoradas também pareciam ter tido dificuldades na maioria das perguntas: por exemplo, os sujeitos #3, #5, #6 e #9 tinham uma alta proporção de respostas incorretas.

Tabela 4.2 – Desempenho da identificação de *smells* dos dois grupos experimentais usando a IDE.

Grupo 1				
Participante	**Respostas Corretas**	**Respostas Incorretas**	**Pulos**	**Tempo Médio**†
#1	3	7	0	2,340
#2	2	8	0	1,080
#3	5	5	0	1,500
#4	3	7	0	1,380
#5	4	6	0	1,140
#6	3	7	0	1,680
#7	2	8	0	1,860
#8	5	5	0	2,220
#9	2	8	0	1,680
Grupo 2				
Participante	**Respostas Corretas**	**Respostas Incorretas**	**Pulos**	**Tempo Médio**†
#1	4	6	0	1,560
#2	3	6	1	1,740
#3	0	6	4	540
#4	1	9	0	2,160
#5	2	2	6	1,080
#6	0	6	4	1,200
#7	2	8	0	1,080
#8	0	7	3	1,740
#9	2	4	4	1,380
Estatística descritiva para ambos os grupos experimentais				
Min	0	2	0	540
Max	5	9	6	2,340
Média	2.39	6.39	1.22	1,520.00
Dev.Padrão	1.54	1.69	3.95	464.30
† O tempo médio é indicado em segundos.				

Fonte: Santos et al. (2019).

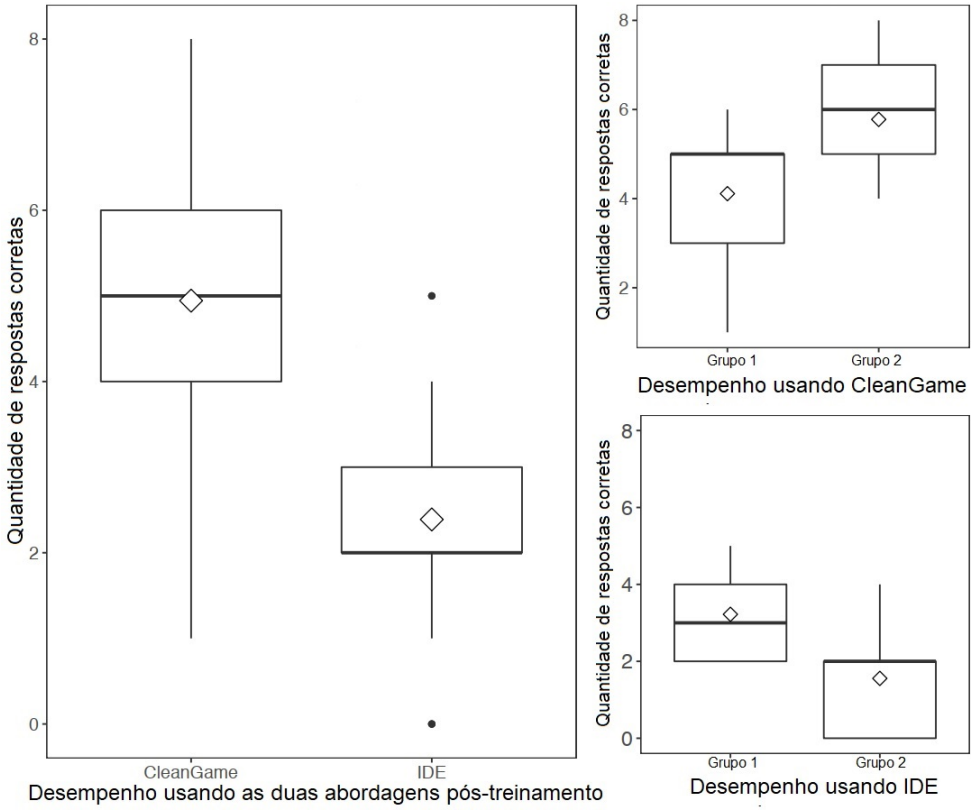
Por outro lado, ao usar o CleanGame, os participantes raramente pularam as perguntas (como mostrado na Tabela 4.1). Diante disso, foi suposto que os participantes tenham menos probabilidade de ignorar perguntas ao usar o CleanGame devido às dicas relacionadas a métricas, refatoração e definição fornecidas pela ferramenta. De acordo com os resultados da Tabela 4.1, o tipo de dica mais comumente solicitado era relacionado à métrica: ao analisar as 10 tarefas de identificação de *code smells*, os participantes solicitaram, em média, aproximadamente cinco dicas relacionadas a métricas.

Curiosamente, as dicas relacionadas à refatoração não foram solicitadas com muita frequência pelos participantes. As dicas relacionadas à definição foram as menos solicitadas. Acredita-se que esses resultados possam indicar que os sujeitos tiveram uma boa compreensão

dos conceitos sobre *code smells*, mas precisavam de algum tipo de métrica para respaldar suas opiniões sobre se estavam ou não olhando para um determinado *smell*.

Dado que a IDE não oferece muito suporte em termos de identificação de *code smells*, os resultados indicam que lidar com essas tarefas em uma IDE parece ser mais difícil para a maioria dos participantes; portanto, os eles parecem parar de responder com mais frequência no momento em que a tarefa de identificação de *smells* fica complicada.

Figura 4.1 – Visão geral do desempenho dos sujeitos experimentais em termos de identificação adequada de *code smells* usando as duas abordagens pós-treinamento.



Fonte: Santos et al. (2019).

4.7.2 Teste de Hipótese

Para testar as hipóteses formuladas na Seção 4.2, foi aplicado um teste de soma e classificação de Wilcoxon [1] emparelhado. Como foram criadas hipóteses, de acordo com os resultados desse teste não paramétrico, os participantes tiveram um desempenho significativamente melhor ao usar o CleanGame do que ao usar uma IDE ($V = 125,5$, $p = 0,003$).

[1] teste de hipóteses não paramétrico utilizado quando se deseja comparar duas amostras relacionadas, amostras emparelhadas ou medidas repetidas em uma única amostra para avaliar se os postos médios populacionais diferem

4.8 Pesquisa Atitudinal

Esta seção descreve os resultados da pesquisa atitudinal realizada para responder $\mathbf{RQ}_2$ e $\mathbf{RQ}_3$. Após desenvolver um rascunho inicial do questionário da pesquisa, foi realizado um teste piloto com um grupo de cinco estudantes de graduação em Ciência da Computação. O objetivo era validar o questionário em termos de clareza, objetividade e correção. O questionário foi refinado com base nos comentários do estudo piloto e então uma versão online foi criada usando o Google Forms [2].

A tabela 4.3 resume cada pergunta no questionário. O questionário foi composto por 22 perguntas, divididas em três partes: *Q1* a *Q4* destinam-se a coletar informações básicas dos participantes; *Q5* a *Q9* são perguntas sobre identificação de *code smells* (com e sem CleanGame); *Q10* ao *Q24* estão relacionadas à experiência dos participantes ao usar o CleanGame. Vale ressaltar que as perguntas Q10 a Q21 foram adaptadas do MEEGA + (PETRI; WANGENHEIM; BORGATTO, 2017), que é uma estrutura para avaliar jogos sérios adaptados ao ensino da computação.

[2] https://www.google.com/forms/about/

Tabela 4.3 – Questionário

ID	Questões	Tipo de resposta
Q1	Nível do estudante	Escolha Única: a. Estudante de graduação de ciência da computação; b. Estudante de graduação em sistemas de informação; c. Graduado em ciência da computação.
Q2	Idade	Escala Nominal: (1) 17 - 22 anos; (2) 23 a 28 anos; (3) 29 a 34 anos; (4) Mais de 34 anos.
Q3	Com que frequência você costuma jogar jogos digitais e não-digitais (de cartas, tabuleiro, etc.)?	Escala Nominal: (1) Nunca; (2) Raramente; (3) as vezes; (4) com uma boa frequencia; (5) diariamente
Q4	Qual sua experiência com java/orientação a objetos?	Escala nominal: (1) Sem experiência; (2) Experiência Acadêmica; (3) Experiência profissional iniciante; (4) Experiência Profissional avançada
Q5	Qual o nível de dificuldade da tarefa realizada sem apoio do CleanGame?	Escala Nominal: (1) Muito Fácil; (2) Fácil; (3) Moderado; (4) Difícil; (5) Muito Difícil.
Q6	Qual o nível de dificuldade da tarefa realizada com apoio da CleanGame?	Nominal Scale: Muito Fácil; (2) Fácil; (3) Moderado; (4) Difícil; (5) Muito Difícil.
Q7	Eu pulei questões porque estava muito difícil.	Escala Likert*
Q8	Eu respondi as questões sem arriscar nenhuma.	Escala Likert*
Q9	Tentei resolver os desafios antes de solicitar as dicas, solicitei apenas quando já havia verificado todo o código e não encontrei o bad smell.	Escala Likert*
Q10	[Desafio] O CleanGame é adequadamente desafiador e não se torna monótono nas suas tarefas (repetitivo ou com tarefas chatas).	Escala Likert*
Q11	[Satisfação] Completar as tarefas do CleanGame me deu um sentimento de realização, foi devido ao meu esforço pessoal que consegui avançar no CleanGame.	Escala Likert*
Q12	[Satisfação] Eu recomendaria o CleanGame para meus colegas.	Escala Likert*
Q13	[Interação Social] O CleanGame promove momentos de competição entre os jogadores.	Escala Likert*
Q14	[Diversão] Houve algum elemento no CleanGame que capturou minha atenção.	Escala Likert*
Q15	[foco]O jogo me manteve motivado na realização das atividades, perdi a noção do tempo, e me esqueci do sobre o ambiente ao meu redor.	Escala Likert*
Q16	[Relevância] O conteúdo do CleanGame é relevante para os meus interesses e é claro como o mesmo está relacionada à disciplina.	Escala Likert*
Q17	[Relevância] Gostaria de utilizar mais ferramentas similares durante minha formação acadêmica.	Escala Likert*
Q18	[Relevance] Eu prefiro exercitar conceitos sobre bad smells com o CleanGame do que de outra forma (outro método de ensino).	Escala Likert*
Q19	[Percepção de aprendizagem] O CleanGame contribuiu para a minha aprendizagem e foi eficiente em comparação com outras atividades da disciplina.	Escala Likert*
Q20	[Percepção de aprendizagem] O CleanGame (módulo Identificação) contribuiu para aplicar conceitos relacionados a identificação de bad smells.	Escala Likert*
Q21	[Percepção de aprendizagem] CleanGame (módulo Quiz) contribuiu para relembrar conceitos sobre bad smells.	Escala Likert*
Q22	Quais os Pontos Positivos do uso do CleanGame?	Pergunta aberta
Q23	Quais os Pontos Negativos do uso do CleanGame?	Pergunta aberta
Q24	Outras considerações	Pergunta aberta

* Likert Scale: (-2) Discordo totalmente; (-1) Discordo; (0) Indiferente; (1) Concordo; (2) Concordo totalmente;

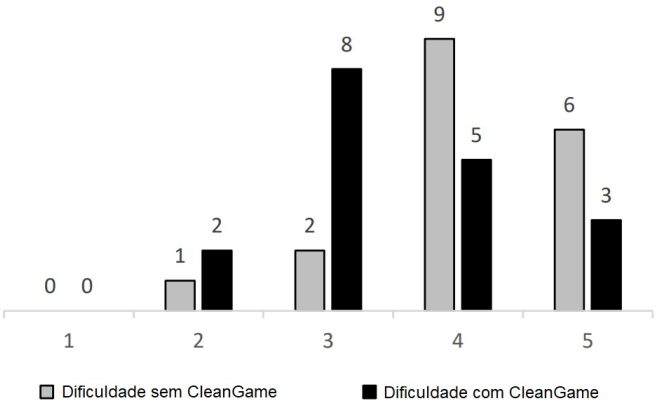
Os participantes foram convidados a responder ao questionário imediatamente ao final do experimento. Foi mencionado de forma clara aos alunos que o preenchimento do mesmo era opcional e anônimo.

4.8.1 Resultados da Pesquisa Atitudinal

Dezoito participantes responderam ao questionário, sendo a maioria dos mesmos (treze participantes, o que representa aproximadamente 72,2 % da nossa amostra) com idade entre 23 e 28 anos. Além disso, treze participantes (72,2 %) afirmaram jogar pelo menos uma vez por mês, dos quais sete (38,9 %) afirmaram jogar diariamente. Apenas três participantes (16,7 %) afirmaram nunca jogar. Em relação à experiência dos participantes com Java ou desenvolvimento orientado a objetos: onze participantes (61,1 %) afirmaram ter experiência profissional com Java ou desenvolvimento orientado a objetos e sete (38,9 %) afirmaram ter apenas experiência acadêmica.

A Figura 4.2 destaca os resultados do questionário sobre as perguntas Q5 e Q6, que se referem a dificuldade de realizar atividades de identificação de *code smells* com e sem o apoio do CleanGame. Os resultados parecem sugerir que os participantes consideraram a atividade mais desafiadora para executar sem o CleanGame.

Figura 4.2 – Dificuldade em executar a identificação de *code smells* sem e com o CleanGame.



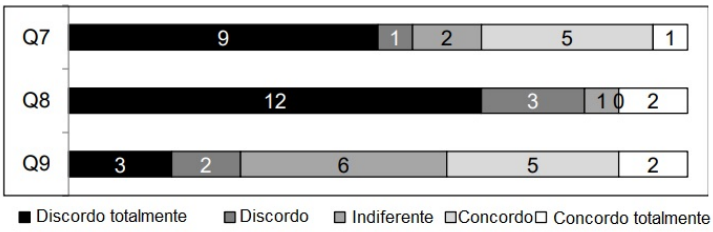
Fonte: Santos et al. (2019).

A Figura 4.3 mostra as respostas coletadas para as perguntas Q7, Q8 e Q9. Observando as respostas do Q7, verifica-se que a maioria dos participantes evitaram pular perguntas, independentemente do nível de dificuldade. Quanto ao Q8, apenas dois participantes (1,1 %) afirmaram que tentaram responder a todas as perguntas conscientemente. Portanto, pode-se presumir que alguns participantes tentaram adivinhar a resposta correta para algumas tarefas de identificação de *code smells*.

Finalmente, os participantes tiveram opiniões contraditórias sobre o Q9: os resultados parecem sugerir que alguns participantes tentaram tirar proveito das dicas fornecidas pelo CleanGame antes de iniciar a tarefa de identificação de *code smells*. Por outro lado, alguns par-

ticipantes só aproveitaram as dicas depois de tentar exaustivamente compreender determinada tarefa.

Figura 4.3 – Resultados das perguntas da pesquisa Q7, Q8 e Q9.



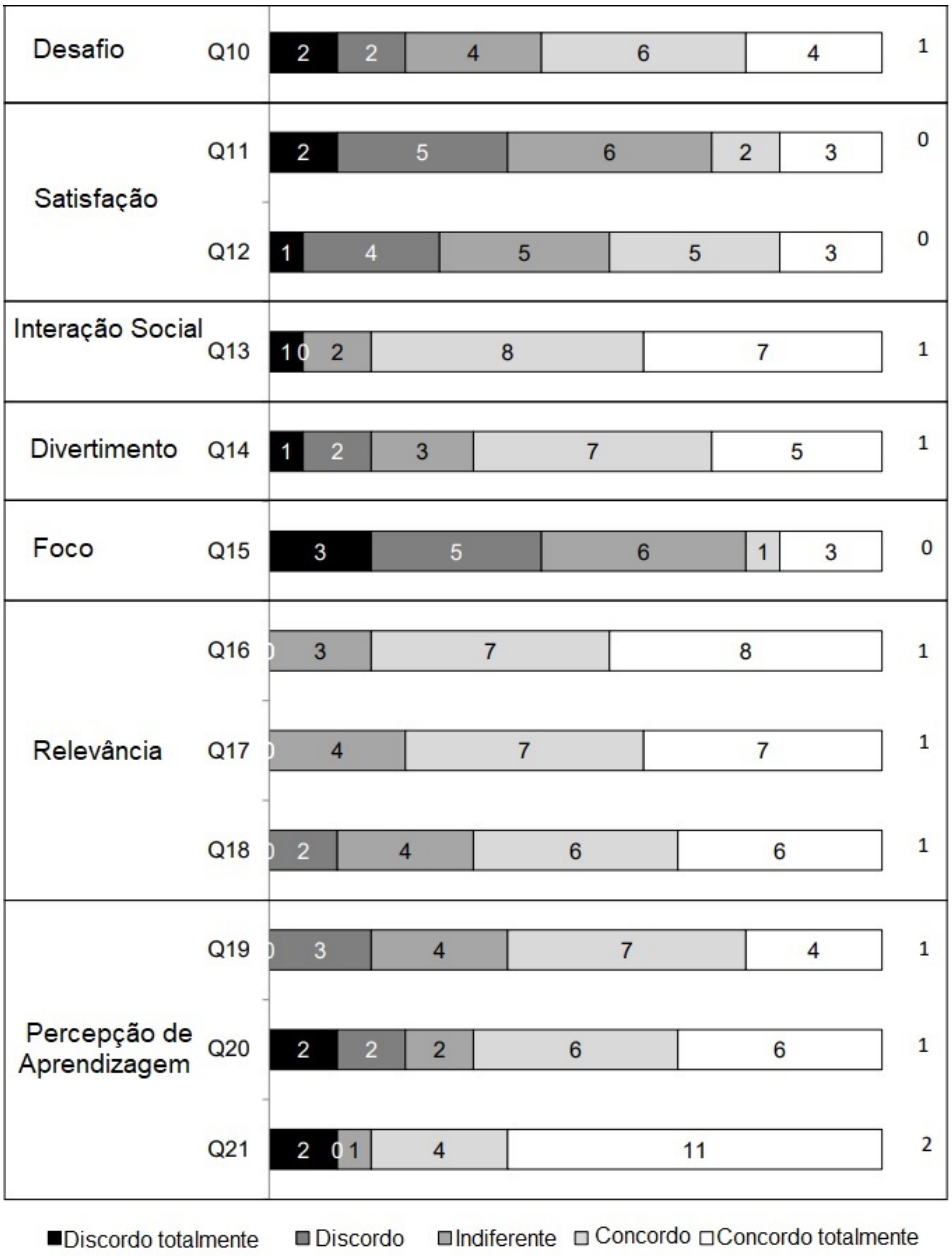
Fonte: Santos et al. (2019).

A Figura 4.4 mostra as respostas relacionadas às experiências dos participantes usando o CleanGame. Os itens são agrupados de acordo com os seguintes fatores: desafio, satisfação, interação social, diversão, foco, relevância e aprendizado. Para cada item, a coluna mais à direita apresenta o valor mediano das respostas dos participantes, variando de 'Discordo totalmente' (-2) a 'Concordo totalmente' (2). Nenhum fator apresentou valor mediano negativo. Os fatores 'satisfação' e 'foco' apresentaram valores medianos de 0, significando que esses aspectos foram observados com indiferença ou opinião mista pelos participantes.

Para todos os outros fatores, os valores medianos foram positivos. Os itens Q14 e Q19 tiveram as respostas 'Concordo totalmente' com mais frequencia. Esses itens estão relacionados à adequação do conteúdo do CleanGame ao curso, e se o quiz ajudou a relembrar conceitos já trabalhados sobre *code smells*. Com exceção dos itens Q11, Q12 e Q15, todos os outros itens receberam mais de 50 % de respostas positivas ('Concordo' e 'Concordo totalmente').

Os itens com maior número de respostas positivas foram: Q13 (probabilidade de recomendar o CleanGame para outros alunos), Q16 (adequação do conteúdo do CleanGame) e Q21 (quanto o CleanGame contribuiu para relembrar os conceitos sobre *code smells*) com 83,3 % de respostas positivas cada. Em contraste, os itens Q15 (foco), Q11 (satisfação) e Q12 (sentimento de conquista) receberam o maior número de respostas negativas, com 44,4 %, 38,9 % e 27,8 % respectivamente, de respostas 'Discordo' ou 'Discordo totalmente'.

Figura 4.4 – Experiência dos participantes com o CleanGame



Fonte: Santos et al. (2019).

Quanto a $\mathbf{RQ}_3$, os comentários positivos e negativos dos participantes foram capturados nas respostas do Q22, Q23 e Q24. Para reunir e sintetizar esse *feedback*, foi empregada uma abordagem inspirada na técnica da teoria do fundamento Stol, Ralph e Fitzgerald (2016). Para isso dois pesquisadores analisaram as respostas individualmente e marcaram segmentos relevantes com ’códigos’ (ou seja, marcação com palavras-chave). Posteriormente, os pesquisadores compararam seus códigos para chegar a um consenso e tentaram agrupar esses códigos em categorias relevantes. Consequentemente, foi possível contar o número de ocorrências de

códigos e o número de itens em cada categoria para entender quais foram os pontos positivos e negativos recorrentes relatados pelos participantes.

As tabelas 4.4 e 4.5 listam os aspectos positivos e negativos relatados. A coluna 'código' agrupa itens recorrentes observados nas respostas, na coluna 'categoria' é listada uma categoria ampla usada para agrupar os códigos, já a coluna 'ocorrências' apresenta o número de vezes que cada código apareceu nas respostas.

Tabela 4.4 – Aspectos positivos declarados pelo participantes

Aspectos positivos	**Categoria**	**#**
Ferramenta lúdica e interativa	Design / Usabilidade	7
Suporte à compreensão de *code smells*	Aprendizagem	6
Competição	Gamificação	5
Facilidade de uso	Design / Usabilidade	3
Dicas	Gamificação	2
Placares dinâmicos	Gamificação	2
Questões de múltipla escolha	Estrutura das perguntas	2
Adequado a diferentes perfis de alunos	Aprendizagem	1
Sistema de pontuação	Gamificação	1
Visualização de perguntas e dicas	Design / Usabilidade	1
Motivador	Aprendizagem	1
Interessante para cursos online	Aprendizagem	1

Fonte: Santos et al. (2019).

Foram encontradas 32 ocorrências de 12 códigos distintos que descrevem aspectos positivos. Esses códigos estão agrupados em quatro categorias: 'Design e usabilidade' (11 ocorrências); 'Gamificação' (10 ocorrências); 'Aprendizado' (9 ocorrências); e 'Estrutura da pergunta' (2 ocorrências). Os códigos mais recorrentes encontrados foram os seguintes: 'Ferramenta lúdica e interativa' (7 ocorrências); 'Suporte à compreensão de *code smells*' (6 ocorrências); e 'Competição' (5 ocorrências). Esses resultados fornecem evidências da atitude positiva dos alunos em relação aos efeitos do CleanGame (e sua abordagem gamificada) quando aplicados na aquisição de habilidades na identificação de *code smells*.

Encontramos 28 ocorrências de 15 códigos distintos, representando *feedback* negativo. Esses códigos estão agrupados em três categorias: 'Design e usabilidade' (11 ocorrências); 'Regras de negócios' (10 ocorrências); e 'Design do experimento' (7 ocorrências). Os problemas no grupo 'Design e usabilidade' estão relacionados à interface gráfica do usuário, como seus elementos visuais são organizados ou à falta de um elemento visual específico. 'Regras de negócios' são problemas relacionados a como as coisas funcionam no software. Esses são os comentários mais interessantes, porque afetam o aspecto funcional do CleanGame. Por exemplo, os códigos de 'Regras de negócios' mais recorrentes estavam relacionados a mostrar a resposta correta como um *feedback* para o usuário depois de escolher uma escolha errada e a sugestão de

não divulgar a pontuação de todos os usuários, alegando que isso pode causar constrangimento, ou minar a motivação dos usuários com desempenho inferior.

Por fim, a categoria 'Design do experimento' agrupa códigos relacionados a reclamações relacionadas à organização do experimento. Por exemplo, dois usuários reclamaram da duração das aulas que ocorreram antes do experimento. Observamos que a maioria dos aspectos negativos são realmente oportunidades de melhoria e não comprometem o processo de aprendizado usando o CleanGame.

Tabela 4.5 – Aspectos negativos declarados pelos participantes

Aspectos negativos	**Categoria**	**#**
Problemas na interface	*Design* / Usabilidade	7
Deve mostrar a resposta correta após falha	Regras de negócios	4
Divulgação de pontuações de todos os participantes	Regras de negócios	4
Tamanho do código	Design do Experimento	2
Sistema de pontuação confuso	Regras de negócios	1
Regras para perder pontuação ao usar dicas	Regras de negócios	1
Deveria ter outros tipos de perguntas	*Design* / Usabilidade	1
Não é possível ver dicas já usadas	Regras de negócios	1
Duração do experimento	Design do Experimento	1
Forma de exibição de pontos ganhos e perdidos	*Design* / Usabilidade	1
Diferença entre a duração do questionário e as atividades de identificação	Design do Experimento	1
Instruções ruins para o experimento	Design do Experimento	1
Deveria ter fornecido as respostas corretas até o final do experimento	Design do Experimento	1
Métricas usadas desconhecidas	Design do Experimento	1
Deveria mostrar a quantidade de perguntas	*Design* / Usabilidade	1

Fonte: Santos et al. (2019).

Quanto a $\mathbf{RQ}_2$, os resultados da pesquisa de atitude parecem sugerir que a maioria dos participantes mostraram uma atitude positiva em relação ao CleanGame. Os resultados do Q5 e Q6 indicam que os participantes acharam menos difícil praticar a identificação de *code smells* com o suporte do CleanGame. Os resultados relacionados à experiência dos participantes com o CleanGame mostraram uma percepção positiva em relação à relevância, percepção da aprendizagem e aspectos de interação social da ferramenta. No entanto, houve algumas opiniões contraditórias sobre foco e satisfação.

Entre os aspectos positivos descritos pelos participantes, houve 10 referências à gamificação e 9 aos efeitos positivos na aprendizagem, dos 32 códigos identificados. Portanto, temos resultados positivos sobre a atitude dos alunos em relação ao CleanGame, especialmente com relação à estratégia de gamificação usada na ferramenta e seu efeito no aprendizado.

4.9 Considerações Finais

Foi avaliado empiricamente se a gamificação pode ter um impacto positivo no reforço pós-treinamento para habilidades e conceitos de identificação de *code smells*. Até onde sabe-se, este é o primeiro estudo que aplica a gamificação para envolver os alunos em tarefas de identificação de *code smells* durante o reforço pós-treinamento. Para avaliar a eficácia da gamificação nesse contexto, foi usada a taxa média de respostas corretas. De acordo com os resultados desse experimento, em média, os sujeitos conseguiram identificar o dobro de *smells* durante o reforço da aprendizagem com a abordagem gamificada em comparação à abordagem orientada a IDE: os resultados de um teste não paramétrico mostraram que os sujeitos têm desempenho significativamente melhor ao usar o CleanGame do que ao usar um IDE. Essas descobertas foram interpretadas como apoio geral à hipótese de que a gamificação pode ser aplicada para envolver os alunos em atividades que tendem a ser um pouco tediosas e complexas, a saber, identificação de *code smells*. Além disso, os sujeitos estavam menos aptos a pular as tarefas de identificação de *smells* ao usar o CleanGame. Acredita-se que isso possa ser atribuído às dicas relacionadas a métricas, refatoração e definição, fornecidas pela ferramenta.

Os resultados da pesquisa de atitudes pós-experimento sugerem que a maioria dos participantes demonstraram uma atitude positiva em relação ao CleanGame (e sua estratégia de gamificação) como uma ferramenta de suporte educacional para a prática da identificação de *code smells*. A avaliação dos participantes quanto a ferramenta revelou que, enquanto 'Foco' e 'Satisfação' foram os aspectos com classificação mais baixa (no entanto, sem classificação negativa), os outros aspectos foram classificados de forma positiva, especialmente 'Relevância', 'Percepção de Aprendizagem' e 'Interação Social'.

5 CONCLUSÃO

Este capítulo aborda os resultados deste trabalho, com relação a seus objetivos, contribuições, limitações e trabalhos futuros. A Seção 5.1 resume as principais conclusões. A Seção 5.2 analisa as principais contribuições. A Seção 5.3 apresenta as limitações desse trabalho. A Seção 5.4 descreve possíveis ideias para trabalhos futuros.

5.1 Síntese

Embora existam pesquisas voltadas à maneiras alternativas de ensino em Engenharia de Software, essa atividade ainda é um desafio. A importância de manter aprendizes motivados no ambiente de trabalho/estudo e a dificuldade da introdução da prática no ensino em Engenharia de Software pode ser notada nos dias atuais. Com a evolução da tecnologia, cada dia mais ela está inserida no cotidiano de todos. Neste sentido, diferentes práticas motivacionais são adotadas. Dessa forma, embora a gamificação seja uma tendência recente, principalmente na área de educação em Engenharia de Software, ela vem se popularizando no contexto educacional.

Neste trabalho, foi proposto e avaliado um sistema gamificado para auxiliar na fixação de conteúdos relacionados a *code smells* de forma prática e de maneira menos tediosa. Para atingir esse objetivo, foram realizados estudos em torno da literatura dos assuntos base para o desenvolvimento do trabalho, para verificar a viabilidade da contribuição para ensino de Engenharia de Software e nortear os passos a serem desenvolvidos. Em seguida, um sistema gamificado voltado para o ensino de *code smells* foi projetado e implementado.

Em relação ao objetivo específico OE_1 deste trabalho ("*Investigar o uso de gamificação em ferramentas e sistemas para apoio ao ensino de Engenharia de Software*"), a revisão bibliográfica apresentada no Capítulo 2 mostrou que existe uma dificuldade entre desenvolvedores de saber quando refatorar um código, ou seja, saber identificar um *code smell*, já que essa não é uma atividade trivial; apresenta que embora existam pesquisas voltadas a métodos de ensino que tornem o ensino em Engenharia de Software menos tedioso e mais prático isso ainda é um desafio. Apresenta também que apesar de a gamificação ser abordada em diversas áreas, na educação em Engenharia de Software ainda está em sua infância. No Capítulo 2.4 pode ser observado que existem trabahos relacionados a gamificação e educação em Engenharia de Software, mas na busca realizada, não foi encontrado na literatura nenhum trabalho que aborde *code smells* da maneira proposta neste trabalho.

Em relação ao OE_2 (*Avaliar ferramentas de identificação de code smells*), a revisão bibliográfica realizada no Seção 2.1.3 mostrou como algumas se comportam e a partir disso foi definido qual a mais pertinente para o trabalho apresentado.

Em relação ao OE_3 (*Implementar o sistema utilizando os conceitos fundamentais da gamificação*), o sistema, CleanGame foi projetado e implementado conforme apresentado no Capítulo 3, a estrutura do CleanGame foi definida de acordo com as diretrizes observadas nos capítulos 2 e 2.4, já que um sistema gamificado, pode não ser bem sucedido por sua má implementação. O sistema foi implementado com os elementos de jogo mais aceitos na literatura e de forma que o mesmo fosse intuitivo e não cansativo independente do nível de conhecimento do usuário sobre *code smells*.

Em relação ao OE_4 (*Avaliar o uso do sistema desenvolvido através de um experimento*), foi realizado um experimento com 18 alunos da Universidade federal de Minas Gerais (UFMG), como apresentado no Capítulo 4. O experimento foi realizado em duas etapas, em que os estudantes eram expostos a uma atividade pós-treinamento, da maneira tradicional e através do CleanGame. Após essas etapas todos os participantes responderam um questionário sobre a experiência do uso do CleanGame. Com isso, foi possível observar uma reação positiva dos alunos em relação ao uso do CleanGame, inclusive de acordo com os resultados, em média, eles conseguiram identificar o dobro de *smells* durante o reforço da aprendizagem com a abordagem gamificada em comparação à abordagem orientada a IDE.

Portanto, os resultados mostram que o uso do CleanGame em atividades de auxílio na fixação e aprendizado de *code smells* é relevante e mais eficaz em comparação com a abordagem tradicional.

5.2 Contribuições

Conforme discutido nos Capítulo 2 e 2.4, a gamificação na área da educação em Engenharia de Software ainda esta em sua infância e métodos que avaliem sua eficacia ainda são necessários, além disso não foi encontrado na literatura trabalhos que envolvam gamificação e aprendizado de *code smells* que abordem códigos reais.

Portanto, este trabalho apresenta o CleanGame, um sistema gamificado para o auxílio no aprendizado e fixação de conteúdos relacionados a *code smells*, de diferentes maneiras, inclusive apresentando códigos reais ao jogador e sua validação, realizada através de um

experimento. Além disso uma metodologia de desenvolvimento do sistema implementado é apresentada, conforme descrito nas Seções 3.1 e 3.2.

5.3 Limitações

O trabalho descrito apresentou limitações quanto ao número reduzido de sua amostra, já que o experimento foi realizado apenas com 18 indivíduos, outro problema relacionado à amostra foi a heterogeneidade da mesma, por possuir alunos de graduação e pós-graduação, fazendo com que o resultado pudesse ser afetado, considerando que os mesmos poderiam ter níveis diferentes de conhecimento.

O módulo *BATALHA* do CleanGame não foi validado no experimento, dessa forma essa é uma outra limitação, já que apenas os módulos *QUIZ* e *IDENTIFICAÇÃO* foram validados no experimento.

O tempo de realização do experimento foi algo mencionado pelos participantes do mesmo, por terem sido executadas três tarefas em um período curto, dessa forma o resultado poderia ter variações, dependendo do tempo disponibilizado. Continuando nos relatos dos participantes o tamanho dos classes também foi uma limitação, já que as mesmas não eram tão curtas, por terem sido coletadas da Landfill (PALOMBA et al., 2015b)[1], para aumentar a confiabilidade das mesmas.

O CleanGame foi validado apenas do ponto de vista dos estudantes, resultados diferentes poderiam ser obtidos caso o experimento fosse realizado com profissionais da área ou do ponto de vista do educador.

5.4 Trabalhos Futuros

Considerando que a primeira tentativa de avaliar o CleanGame foi positiva, esse resultado é apenas um indicativo com uma amostra pequena, dessa forma, é necessário expandir o escopo da avaliação, com amostras maiores. Um outro ponto a avaliar é o módulo Batalha já que ele não foi avaliado no experimento apresentado nesse trabalho.

Além disso, embora as Seções 3.1 e 3.2 apresentem a metodologia utilizada para desenvolvimento do sistema, espera-se explanar de forma mais clara e com maiores detalhes como

[1] plataforma baseada na Web para compartilhar e validar conjuntos de dados de *code smell*

gamificar um sistema, para que essa possa ser utilizada como referência para gamificar sistemas de diferentes áreas de ensino.

Dessa forma, com a metodologia descrita de forma mais clara, é possível criar um *framework* genérico para ser utilizado em diversas áreas da Engenharia de Software, tais como: requisitos de software, teste de software, gerência de projetos, qualidade de software, *design* e construção de software, entre outros.

Por fim, almeja-se realizar uma avaliação do ponto de vista do educador, já que o sistema é implementado para importar classes diretas do *github* e serem validadas pelo mesmo, com isso, poderia ser feito um experimento orientado durante determinado tempo, para observar a evolução dos alunos. Continuando no parte da análise do sistema o mesmo pode ser validado do ponto de vista de profissionais, para ver qual sua pertinência no ambiente profissional.

REFERÊNCIAS

AKPOLAT, B. S.; SLANY, W. Enhancing software engineering student team engagement in a high-intensity extreme programming course using gamification. In: IEEE. **2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE&T)**. [S.l.], 2014. p. 149–153.

AL-QUTAISH, R. E. Quality models in software engineering literature: an analytical and comparative study. **Journal of American Science**, v. 6, n. 3, p. 166–175, 2010.

ALHAMMAD, M. M.; MORENO, A. M. Gamification in software engineering education: A systematic mapping. **Journal of Systems and Software**, v. 141, p. 131 – 150, 2018. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121218300645>.

ALVES, F. P. et al. A rede social móvel foursquare: uma análise dos elementos de gamificação sob a ótica dos usuários. In: **Workshop Proc. WAIHCWS**. [S.l.: s.n.], 2012. v. 2012.

ALVES, R. A.; MARTINS, R. C.; PAULISTA, P. H. Operação e manutenção de software: Uma abordagem teórica. **Revista Univap**, v. 22, n. 40, p. 766, 2017.

AMMAR, B. B.; BHIRI, M. T. Pattern-based model refactoring for the introduction association relationship. **Journal of King Saud University-Computer and Information Sciences**, Elsevier, v. 27, n. 2, p. 170–180, 2015.

AMORIM, L. B. V. d. et al. Um método para descoberta automática de regras para a detecção de bad smells. Universidade Federal de Alagoas, 2014.

ANDERSON, P. E.; NASH, T.; MCCAULEY, R. Facilitating programming success in data science courses through gamified scaffolding and learn2mine. In: **Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education**. New York, NY, USA: ACM, 2015. (ITiCSE '15), p. 99–104. ISBN 978-1-4503-3440-2. Disponível em: <http://doi.acm.org/10.1145/2729094.2742597>.

ARCOVERDE, R.; GARCIA, A.; FIGUEIREDO, E. Understanding the longevity of code smells: preliminary results of an explanatory survey. In: ACM. **Proceedings of the 4th Workshop on Refactoring Tools**. [S.l.], 2011. p. 33–36.

ARTAZA, H. et al. Top 10 metrics for life science software good practices. **F1000Research**, Faculty of 1000 Ltd, v. 5, 2016.

BARROS, V. P. A. **Um método para a refatoração de software baseado em frameworks de domínio**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2015.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software architecture in practice**. [S.l.]: Addison-Wesley Professional, 2003.

BASTARRICA, M. C.; PEROVICH, D.; SAMARY, M. M. What can students get from a software engineering capstone course? In: **2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)**. [S.l.: s.n.], 2017. p. 137–145.

BAVOTA, G. et al. An experimental investigation on the innate relationship between quality and refactoring. **Journal of Systems and Software**, Elsevier, v. 107, p. 1–14, 2015.

BELL, J.; SHETH, S.; KAISER, G. Secret ninja testing with halo software engineering. In: ACM. **Proceedings of the 4th international workshop on Social software engineering**. [S.l.], 2011. p. 43–47.

BERKLING, K.; THOMAS, C. Gamification of a software engineering course and a detailed analysis of the factors that lead to it's failure. In: IEEE. **2013 International Conference on Interactive Collaborative Learning (ICL)**. [S.l.], 2013. p. 525–530.

BOAS, J. L. V. et al. Um werb service para gamificação. In: **VIII Simposio Internacional de Informática Educativa, SIEE**. [S.l.: s.n.], 2016. p. 123–128.

BORGES, S. S. et al. A systematic mapping on gamification applied to education. In: **Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14**. [S.l.]: ACM Press, 2014. p. 216–222.

BORRAS-GENE, O.; MARTINEZ-NUNEZ, M.; FIDALGO-BLANCO, Á. New challenges for the motivation and learning in engineering education using gamification in mooc. **International Journal of Engineering Education**, v. 32, n. 1, p. 501–512, 2016.

BROWN, N. et al. Managing technical debt in software-reliant systems. In: ACM. **Proceedings of the FSE/SDP workshop on Future of software engineering research**. [S.l.], 2010. p. 47–52.

BUISMAN, A. L.; EEKELEN, M. C. van. Gamification in educational software development. In: **CSERC**. [S.l.: s.n.], 2014. p. 9–20.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. **IEEE Transactions on software engineering**, IEEE, v. 20, n. 6, p. 476–493, 1994.

CHUG, A.; GUPTA, M. A quality enhancement through defect reduction using refactoring operation. In: IEEE. **Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on**. [S.l.], 2017. p. 1869–1875.

CUNNINGHAM, W. The wycash portfolio management system. In: **Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)**. New York, NY, USA: ACM, 1992. (OOPSLA '92), p. 29–30. ISBN 0-89791-610-7. Disponível em: <http://doi.acm.org/10.1145/157709.157715>.

DETERDING, S. et al. From game design elements to gamefulness: defining gamification. In: ACM. **Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments**. [S.l.], 2011. p. 9–15.

DETERDING, S. et al. **Gamification: Toward a definition**. 2011. 12-15 p.

DEVADIGA, N. M. Software engineering education: Converging with the startup industry. In: **2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)**. [S.l.: s.n.], 2017. p. 192–196.

DHARMAWAN, T.; ROCHIMAH, S. Systematic literature review: Model refactoring. In: IEEE. **Computer Applications and Information Processing Technology (CAIPT), 2017 4th International Conference on**. [S.l.], 2017. p. 1–5.

DUBOIS, D. J.; TAMBURRELLI, G. Understanding gamification mechanisms for software development. In: ACM. **Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering**. [S.l.], 2013. p. 659–662.

FERNANDES, F.; WERNER, C. Towards immersive software engineering education. In: SBC. **Anais do XXI Symposium on Virtual and Augmented Reality**. [S.l.], 2019. p. 7–8.

FIGUEIREDO, E. et al. Um jogo para o ensino de engenharia de software centrado na perspectiva de evolução. In: **Workshop sobre Educação em Computação (WEI–2007)**. [S.l.: s.n.], 2007. v. 15, p. 37–46.

FONTANA, F. A.; BRAIONE, P.; ZANONI, M. Automatic detection of bad smells in code: An experimental assessment. **Journal of Object Technology**, v. 11, n. 2, p. 5–1, 2012.

FONTANA, F. A. et al. Investigating the impact of code smells on system's quality: An empirical study on systems of different application domains. In: IEEE. **Software Maintenance (ICSM), 2013 29th IEEE International Conference on**. [S.l.], 2013. p. 260–269.

FOWLER, M.; BECK, K. **Refactoring: improving the design of existing code**. [S.l.]: Addison-Wesley Professional, 1999.

FREITAS, S. A. A. et al. Gamification in education: A methodology to identify student. In: IEEE. **2017 IEEE Frontiers in Education Conference (FIE)**. [S.l.], 2017. p. 1–8.

FU, S.; SHEN, B. Code bad smell detection through evolutionary data mining. In: IEEE. **Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on**. [S.l.], 2015. p. 1–9.

FU, Y.; CLARKE, P. Gamification based cyber enabled learning environment of software testing. **submitted to the 123rd American Society for Engineering Education (ASEE)-Software Engineering Constituent**, 2016.

GARCÍA, F. et al. A framework for gamification in software engineering. **Journal of Systems and Software**, Elsevier, v. 132, p. 21–40, 2017.

GARCIA, J. et al. Identifying architectural bad smells. In: IEEE. **Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on**. [S.l.], 2009. p. 255–258.

GARCíA, F. et al. A framework for gamification in software engineering. **Journal of Systems and Software**, v. 132, p. 21 – 40, 2017. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121217301218>.

GUO, X.; SHI, C.; JIANG, H. Deep semantic-based feature envy identification. In: **Proceedings of the 11th Asia-Pacific Symposium on Internetware**. New York, NY, USA: ACM, 2019. (Internetware '19), p. 19:1–19:6. ISBN 978-1-4503-7701-0. Disponível em: <http://doi.acm.org/10.1145/3361242.3361257>.

IEEE,ACM. **Computer Engineering Curricula 2016. CE2016. Curriculum Guidelnes for Undergraduate Degree Programas in Computer Engineering.** Association for computing machinery (acm)/ieee computer society. New York, 2016.

ITO, Y. et al. A method for detecting bad smells and its application to software engineering education. In: IEEE. **Advanced Applied Informatics (IIAIAAI), 2014 IIAI 3rd International Conference on**. [S.l.], 2014. p. 670–675.

KAUR, A. A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes. **Archives of Computational Methods in Engineering**, Springer, p. 1–30, 2019.

KAUR, G.; SINGH, B. Improving the quality of software by refactoring. In: IEEE. **Intelligent Computing and Control Systems (ICICCS), 2017 International Conference on**. [S.l.], 2017. p. 185–191.

KHALEEL, F. L. et al. Gamification elements for learning applications. **International Journal on Advanced Science, Engineering and Information Technology**, v. 6, n. 6, p. 868–874, 2016.

KHOMH, F. et al. A bayesian approach for the detection of code and design smells. In: IEEE. **Quality Software, 2009. QSIC'09. 9th International Conference on**. [S.l.], 2009. p. 305–314.

KIM, J. T.; LEE, W.-H. Dynamical model for gamification of learning (dmgl). **Multimedia Tools and Applications**, v. 74, n. 19, p. 8483–8493, Oct 2015. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-013-1612-8>.

KOIVISTO, J.; HAMARI, J. The rise of motivational information systems: A review of gamification research. **International Journal of Information Management**, v. 45, p. 191 – 210, 2019. ISSN 0268-4012. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0268401217305169>.

KOSA, M. et al. Software engineering education and games: a systematic literature review. **Journal of Universal Computer Science**, Technische Universitaet Graz* Institut fuer Informationssysteme und Computer Medien, v. 22, n. 12, p. 1558–1574, 2016.

KRUCHTEN, P. B. The 4+ 1 view model of architecture. **IEEE software**, IEEE, v. 12, n. 6, p. 42–50, 1995.

LASKOWSKI, M. Implementing gamification techniques into university study path-a case study. In: IEEE. **2015 IEEE Global Engineering Education Conference (EDUCON)**. [S.l.], 2015. p. 582–586.

LAURENT, T. et al. Towards a gamified equivalent mutants detection platform. In: IEEE. **Software Testing, Verification and Validation Workshops (ICSTW), 2017 IEEE International Conference on**. [S.l.], 2017. p. 382–384.

LEGAKI, N. Z. et al. Gamification of the future: An experiment on gamifying education of forecasting. In: **Proceedings of the 52nd Hawaii International Conference on System Sciences**. [S.l.: s.n.], 2019.

LIU, H. et al. Identifying renaming opportunities by expanding conducted rename refactorings. **IEEE Transactions on Software Engineering**, IEEE, v. 41, n. 9, p. 887–900, 2015.

MAIGA, A. et al. Support vector machines for anti-pattern detection. In: IEEE. **Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on**. [S.l.], 2012. p. 278–281.

MAJURI, J.; KOIVISTO, J.; HAMARI, J. Gamification of education and learning: A review of empirical literature. In: CEUR-WS. **Proceedings of the 2nd International GamiFIN Conference, GamiFIN 2018**. [S.l.], 2018.

MALHOTRA, R.; CHUG, A. An empirical study to assess the effects of refactoring on software maintainability. In: IEEE. **Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on**. [S.l.], 2016. p. 110–117.

MANSOOR, U. et al. Multi-objective code-smells detection using good and bad design examples. **Software Quality Journal**, Springer, v. 25, n. 2, p. 529–552, 2017.

MARINESCU, R. Detection strategies: Metrics-based rules for detecting design flaws. In: IEEE. **Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on**. [S.l.], 2004. p. 350–359.

MARINESCU, R. Measurement and quality in object-oriented design. In: IEEE. **Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on**. [S.l.], 2005. p. 701–704.

MOHA, N. et al. Decor: A method for the specification and detection of code and design smells. **IEEE Transactions on Software Engineering (TSE)**, IEEE Computer Society, v. 36, n. 1, p. 20–36, 2010.

MORSCHHEUSER, B. et al. How to design gamification? a method for engineering gamified software. **Information and Software Technology**, Elsevier, 2017.

MORSCHHEUSER, B. et al. How to design gamification? a method for engineering gamified software. **Information and Software Technology**, v. 95, p. 219 – 237, 2018. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S095058491730349X>.

MUNRO, M. J. Product metrics for automatic identification of"bad smell"design problems in java source-code. In: IEEE. **Software Metrics, 2005. 11th IEEE International Symposium**. [S.l.], 2005. p. 15–15.

MURPHY-HILL, E.; BLACK, A. P. Refactoring tools: Fitness for purpose. **IEEE software**, IEEE, v. 25, n. 5, 2008.

NASCIMENTO, R.; SANT'ANNA, C. Investigating the relationship between bad smells and bugs in software systems. In: ACM. **Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse**. [S.l.], 2017. p. 4.

OPDYKE, W. F. Refactoring: An aid in designing application frameworks and evolving object-oriented systems. In: **Proc. SOOPPA'90: Symposium on Object-Oriented Programming Emphasizing Practical Applications**. [S.l.: s.n.], 1990.

OUNI, A. et al. Web service antipatterns detection using genetic programming. In: ACM. **Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation**. [S.l.], 2015. p. 1351–1358.

PAIVA, T. et al. On the evaluation of code smells and detection tools. **Journal of Software Engineering Research and Development**, Springer, v. 5, n. 1, p. 7, 2017.

PAIVA, T. et al. Experimental evaluation of code smell detection tools. 2015.

PALOMBA, F. et al. Do they really smell bad? a study on developers' perception of bad code smells. In: IEEE. **Software maintenance and evolution (ICSME), 2014 IEEE international conference on**. [S.l.], 2014. p. 101–110.

PALOMBA, F. et al. Mining version histories for detecting code smells. **IEEE Transactions on Software Engineering**, IEEE, v. 41, n. 5, p. 462–489, 2015.

PALOMBA, F. et al. Landfill: An open dataset of code smells with public evaluation. In: **2015 IEEE/ACM 12th Working Conference on Mining Software Repositories**. [S.l.: s.n.], 2015. p. 482–485.

PAN, W.-F.; JIANG, B.; LI, B. Refactoring software packages via community detection in complex software networks. **International Journal of Automation and Computing**, v. 10, n. 2, p. 157–166, Apr 2013. ISSN 1751-8520. Disponível em: <https://doi.org/10.1007/s11633-013-0708-y>.

PETERS, R.; ZAIDMAN, A. Evaluating the lifespan of code smells using software repository mining. In: IEEE. **Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on**. [S.l.], 2012. p. 411–416.

PETRI, G.; WANGENHEIM, C. G. von; BORGATTO, A. F. A large-scale evaluation of a model for the evaluation of games for teaching software engineering. In: IEEE. **2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)**. [S.l.], 2017. p. 180–189.

POFFO, M. et al. Gamificação: Agente motivador na aprendizagem de engenharia de software. **Anais do Computer on the Beach**, p. 110–119, 2017.

PRESSMAN, S. R. Engenharia de software: Uma abordagem profissional. 7ª. **Edição. Rio de Janeiro–RJ. Editora McGraw-Hill**, 2011.

QU, W.-Q. et al. Research on teaching gamification of software engineering. In: IEEE. **2014 9th International Conference on Computer Science & Education**. [S.l.], 2014. p. 855–860.

RAAB, F. Codesmellexplorer: Tangible exploration of code smells and refactorings. In: IEEE. **Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on**. [S.l.], 2012. p. 261–262.

REHMAN, F. ur et al. Scrum software maintenance model: Efficient software maintenance in agile methodology. In: IEEE. **2018 21st Saudi Computer Society National Computer Conference (NCC)**. [S.l.], 2018. p. 1–5.

ROJAS, J. M.; FRASER, G. Code defenders: a mutation testing game. In: IEEE. **Software Testing, Verification and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on**. [S.l.], 2016. p. 162–167.

SANTOS, H. M. dos et al. Cleangame: Gamifying the identification of code smells. In: **Proceedings of the XXXIII Brazilian Symposium on Software Engineering**. New York, NY, USA: ACM, 2019. (SBES 2019), p. 437–446. ISBN 978-1-4503-7651-8. Disponível em: <http://doi.acm.org/10.1145/3350768.3352490>.

SASSO, T. D. et al. How to gamify software engineering. In: IEEE. **Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on**. [S.l.], 2017. p. 261–271.

SAVI, R.; WANGENHEIM, C.; BORGATTO, A. Um modelo de avaliação de jogos educacionais na engenharia de software. **Anais do XXV Simpósio Brasileiro de Engenharia de Software (SBES 2011), São Paulo**, 2011.

SHARMA, T.; SPINELLIS, D. A survey on software smells. **Journal of Systems and Software**, Elsevier, v. 138, p. 158–173, 2018.

SINGER, L.; SCHNEIDER, K. It was a bit of a race: Gamification of version control. In: IEEE. **2012 Second International Workshop on Games and Software Engineering: Realizing User Engagement with Game Engineering Techniques (GAS)**. [S.l.], 2012. p. 5–8.

SINGH, N.; SINGH, P. How do code refactoring activities impact software developers' sentiments?-an empirical investigation into github commits. In: IEEE. **Asia-Pacific Software Engineering Conference (APSEC), 2017 24th**. [S.l.], 2017. p. 648–653.

SOUZA, M. R. A. et al. A systematic mapping study on game-related methods for software engineering education. **Information and Software Technology**, v. 95, p. 201–218, 2018. ISSN 0950-5849.

SOUZA, M. R. de A. et al. Gamification in software engineering education: An empirical study. In: IEEE. **Software Engineering Education and Training (CSEE&T), 2017 IEEE 30th Conference on**. [S.l.], 2017. p. 276–284.

SOUZA, M. R. de A. et al. A systematic mapping study on game-related methods for software engineering education. **Information and software technology**, Elsevier, v. 95, p. 201–218, 2018.

SOUZA, P. P. et al. Applying software metric thresholds for detection of bad smells. In: ACM. **Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse**. [S.l.], 2017. p. 6.

SOUZA Érica R.; SOUTO, E. Utilizacao de heuristicas de jogos para avaliacao de um aplicativo gamificado. **Proceedings of SBGames**, p. 666–673, 2015.

SPINELLIS, D. Refactoring on the cheap. **IEEE Software**, v. 29, n. 1, p. 96–95, 2012.

SRIVISUT, K.; MUENCHAISRI, P. Bad-smell metrics for aspect-oriented software. In: IEEE. **Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on**. [S.l.], 2007. p. 1060–1065.

STEGEMAN, M.; BARENDSEN, E.; SMETSERS, S. Towards an empirically validated model for assessment of code quality. In: ACM. **Proceedings of the 14th Koli Calling international conference on computing education research**. [S.l.], 2014. p. 99–108.

STOL, K.-J.; RALPH, P.; FITZGERALD, B. Grounded theory in software engineering research: a critical review and guidelines. In: IEEE. **2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)**. [S.l.], 2016. p. 120–131.

SU, C.-H. The effects of students' motivation, cognitive load and learning anxiety in gamification software engineering education: a structural equation modeling study. **Multimedia Tools and Applications**, v. 75, n. 16, p. 10013–10036, Aug 2016. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-015-2799-7>.

SURYANARAYANA, G.; SAMARTHYAM, G.; SHARMA, T. **Refactoring for software design smells: managing technical debt**. [S.l.]: Morgan Kaufmann, 2014.

THOMAS, C.; BERKLING, K. Redesign of a gamified software engineering course. In: IEEE. **2013 International Conference on Interactive Collaborative Learning (ICL)**. [S.l.], 2013. p. 778–786.

TU, Q.; GODFREY, M. W. The build-time software architecture view. In: **Proceedings IEEE International Conference on Software Maintenance. ICSM 2001**. [S.l.: s.n.], 2001. p. 398–407.

TUFANO, M. et al. When and why your code starts to smell bad (and whether the smells go away). **IEEE Transactions on Software Engineering**, IEEE, v. 43, n. 11, p. 1063–1088, 2017.

USKOV, V.; SEKAR, B. Gamification of software engineering curriculum. In: IEEE. **Frontiers in Education Conference (FIE), 2014 IEEE**. [S.l.], 2014. p. 1–8.

USKOV, V.; SEKAR, B. Gamification of software engineering curriculum. In: IEEE. **2014 IEEE Frontiers in Education Conference (FIE) Proceedings**. [S.l.], 2014. p. 1–8.

VIDAL, S. A.; MARCOS, C.; DÍAZ-PACE, J. A. An approach to prioritize code smells for refactoring. **Automated Software Engineering**, Springer, v. 23, n. 3, p. 501–532, 2016.

VIGGIATO, M.; OLIVEIRA, J.; FIGUEIREDO, E. On the investigation of domain-sensitive bad smells in information systems. In: **13th Brazilian Symposium of Information Systems (SBSI)**. [S.l.: s.n.], 2017. p. 388–395.

WANGENHEIM, C. G. V.; KOCHANSKI, D.; SAVI, R. Revisão sistemática sobre avaliação de jogos voltados para aprendizagem de engenharia de software no brasil. **FEES: Fórum de Educação em Engenharia de Software**, 2009.

WOHLIN, C. et al. **Experimentation in Software Engineering**. [S.l.]: Springer, 2012. 236 p.

ZORZO, A. F. et al. **Referenciais de Formação para os Cursos de Graduação em Computação 2017**. [S.l.]: SBC - Sociedade Brasileira de Computação, 2017.