Pós-Graduação em Ciência da Computação

Gabriel Ibson de Souza

Gamificando um sistema existente: o estudo de caso VazaZika

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

Gabriel Ibson de Souza

Gamificando um sistema existente: o estudo de caso VazaZika

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Engenharia de Software
Orientador: Leopoldo Motta Teixeira

Recife
2019

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S729g Souza, Gabriel Ibson de
Gamificando um sistema existente: o estudo de caso VazaZika / Gabriel Ibson de Souza. – 2019.
83 f.: il., fig., tab.

Orientador: Leopoldo Motta Teixeira.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2019.
Inclui referências.

1. Engenharia de software. 2. Gamificação. 3. Arquitetura de software. I. Teixeira, Leopoldo Motta (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2019-077

Gabriel Ibson de Souza

“**Gamificando um sistema existente: o estudo de caso VazaZika”**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 22/02/2019.

BANCA EXAMINADORA

_____

Prof. Dr. Kiev Santos da Gama
Centro de Informática/UFPE

_____

Prof. Dr. Baldoino Fonseca dos Santos Neto
Instituto de Computação/UFAL

_____

Prof. Dr. Leopoldo Motta Teixeira
Centro de Informática/UFPE
(**Orientador**)

Decido este trabalho ao meu avô José Rodrigues. Se estivesse comigo hoje, tenho certeza que ficaria muito orgulhoso.

AGRADECIMENTOS

Muitas pessoas fizeram parte dessa jornada junto comigo, para que fosse possível chegar ao final. A elas dirijo minha mais profunda gratidão, pois não conseguiria ir tão longe sozinho.

Meu primeiro agradecimento vai para Deus, pois reconheço que sem Ele eu não seria absolutamente nada.

Agradeço imensamente a minha esposa Pâmela Beatriz, por ter sido sempre minha companheira em todas as horas, por me ajudar a chegar até aqui e por sempre me incentivar a ser a melhor versão de mim mesmo.

Agradeço também aos meus pais, por sempre me apoiarem e me incentivarem a crescer como pessoa e profissional.

Agradeço de todo coração ao meu orientador Leopoldo Teixeira. Desde o primeiro dia me fez acreditar que eu poderia ter sucesso. Nunca esquecerei toda sua dedicação e atenção a mim dispensadas.

Agradeço pela oportunidade de ter participado do workshop do VazaZika em 2018. A experiência foi um divisor de águas para que eu pudesse especificar o contexto deste trabalho. Na ocasião, quero lembrar da reciptividade de toda a equipe do projeto, mas em especial aos integrantes da UFAL, primeiramente na pessoa do professor Baldoino, que fez de tudo para que eu me sentisse à vontade durante minha primeira experiência de apresentação em inglês. Em segundo lugar, na pessoa do professor Márcio, que recebeu a mim e ao professor Leopoldo no aeroporto, nos oferecendo cordialmente carona até a pousada, na qual iríamos nos hospedar.

Quero agradecer também aos envolvidos no projeto VazaZika, tanto a equipe da UFAL, quanto da PUC-Rio, por me darem a oportunidade de contribuir e também de realizar essa pesquisa, fornecendo-me as informações necessárias com muita presteza.

RESUMO

Gamificação é uma técnica de usar elementos de jogos em contextos que não são jogos, com o objetivo de aumentar o nível de engajamento dos usuários de determinado sistema. Esta técnica tem sido estudada em diversos contextos de seu uso na indústria. Porém, é possível perceber que os sistemas gamificados, normalmente, são pensados desde o início para atender a requisitos de gamificação. Portanto, é difícil encontrar na indústria e na literatura, exemplos de sistemas existentes que adotaram características de gamificação posteriormente. Com isto, o VazaZika se apresenta como um bom caso a ser estudado, por se tratar de um sistema resultante do processo de gamificar um sistema existente, o VazaDengue. Este sistema representa uma plataforma de software desenvolvida para permitir aos cidadãos colaborarem com os agentes de saúde no combate ao mosquito transmissor da dengue, zika e chikungunya, reportando possíveis pontos de focos do mosquito. Ao ser analisado, o processo de gamificar a plataforma pôde fornecer resultados que contribuem para que futuros pesquisadores e praticantes possam aumentar suas chances de sucesso na implementação de gamificação em sistemas e plataformas atualmente em uso, sobretudo nos aspectos de retirar lições aprendidas a partir da forma pela qual a gamificação foi introduzida na plataforma sob análise. Para isso, foi necessário entender de que forma estava consolidada a arquitetura do então VazaDengue e como a introdução dos elementos de jogos afetaram esta arquitetura, de modo a contribuir para a manutenção ou degradação de sua qualidade. Em um estudo anterior, foram levantados desafios enfrentados durante o processo de gamificação da plataforma relatados sob o olhar dos desenvolvedores. A análise e avaliação do processo de gamificação adotado foram feitas através de métricas que avaliam a estrutura arquitetural de um software em termos de níveis de acoplamento e coesão, tais como o número de dependências, a média de dependências por classe, o número de classes envolvidos em ciclos, e o número total de ciclos do sistema. Com base nestas métricas, contrastamos os resultados com os desafios e decisões tomadas por eles refletidas na arquitetura resultante. Após a análise, foi possível perceber que a qualidade da arquitetura preexistente sofreu impactos negativos ao longo do processo de gamificação, como por exemplo, o aumento significativo de classes envolvidas em ciclos, o que aumentou os níveis de acoplamento do sistema, e que os desafios relatados pelos desenvolvedores, em parte, não correspondem ao que de fato foi implementado por eles na nova arquitetura. Finalmente, algumas sugestões foram dadas em relação a como o processo de gamificação poderia ter causado menos impactos negativos à arquitetura.

Palavras-chaves: Gamificação. Desafios da Gamificação. VazaZika. Arquitetura de Software.

ABSTRACT

Gamification is a technique of using game elements in contexts that are not games, in order to increase the level of engagement of users of a particular system. This technique has been studied in several contexts of its use in industry. However, it is possible to realize that the gamified systems are usually thought from the outset to meet gamification requirements. Therefore, it is difficult to find in industry and literature examples of existing systems that have subsequently adopted gamification features. With this, VazaZika presents itself as a good case to be studied, because it is a system resulting from the process of gamifying an existing system, the VazaDengue. This system represents a software platform developed to allow citizens to collaborate with health agents in the fight against mosquitoes that transmit dengue, zika and chikungunya, reporting possible mosquito outbreaks. When analyzed, the process of gamifying the platform could provide results that contribute to future researchers and practitioners to increase their chances of success in the implementation of gamification in systems and platforms currently in use, especially in the aspects of withdrawing lessons learned from the way by which the gamification was introduced in the platform under analysis. For this, it was necessary to understand how the architecture of the then VazaDengue was consolidated and how the introduction of game elements affected this architecture, in order to contribute to the maintenance or degradation of its quality. In an earlier study, challenges faced were raised during the platform gamification process reported under the developers' view. The analysis and evaluation of the adopted gamification process were done through metrics that evaluate the architectural structure of a software in terms of levels of coupling and cohesion, such as the number of dependencies, the average number of dependencies per class, the number of classes involved in cycles, and the total number of system cycles. Based on these metrics, we contrast the results with the challenges and decisions taken by the developers reflected in the resulting architecture. After the analysis, it was possible to perceive that the quality of the preexisting architecture suffered negative impacts during the process of gamification, as an example, the significant increase of classes involved in cycles, which increased the levels of coupling of the system, and that the challenges reported by the developers, in part, do not correspond to what was actually implemented by them in the new architecture. Finally, some suggestions were given as to how the gamification process could have caused fewer negative impacts to the architecture.

Keywords: Gamification. Gamification Challenges. VazaZika. Software Architecture.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACD	Average Component Dependencies
API	Application Programming Interface
CCD	Cumulative Component Dependencies
DSL	Domain Specific Language
GT	Grounded Theory
HCI	Human-Computer Interaction
IDE	Integrated Development Environment
LOC	Lines of Code
MDA	Mechanics, Dynamics and Aesthetics
STAN	Structure Analysis for Java
UCD	User Centered Design

SUMÁRIO

1 INTRODUÇÃO

Gamificação é “uma maneira divertida de engajar usuários, integrando elementos de jogos em contextos que não são jogos” (DETERDING et al., 2011a). Esta técnica já vem sendo usada com sucesso em diversos domínios, tais como educação, prática de esportes e turismo. No entanto, sistemas gamificados, normalmente, são construídos desde o início atendendo aos requisitos de gamificação, sendo portanto, difícil encontrar sistemas que foram gamificados após sua implementação.

Diante deste cenário, encontramos o VazaZika, uma plataforma de software desenvolvida para atender a necessidade de gamificar um sistema já existente, o VazaDengue. Em 2015 a plataforma VazaDengue foi lançada como um projeto fruto da parceria entre a UFAL (Universidade Federal de Alagoas), PUC-Rio (Pontifícia Universidade Católica do Rio de Janeiro) e Newcastle University com o intuito de disponibilizar um sistema de software capaz de fornecer aos cidadãos meios de colaborar com os agentes de saúde na identificação e denúncia de pontos de focos do mosquito transmissor da dengue, zika e chikungunya.

A plataforma era composta por uma arquitetura orientada a serviços (ERL, 2005), a qual tinha como objetivo fornecer, de maneira facilitada, aos cidadãos, formas de realizar essas denúncias, facilitando assim o trabalho dos agentes na rápida identificação de locais com necessidades de urgente atendimento. A plataforma foi distribuída através de uma aplicação móvel e outra Web, ambas integradas em termos de serviços e bancos de dados, as quais coletavam não somente as denúncias de pontos de foco do mosquito, como também casos de doenças reportados pelos cidadãos.

A necessidade de gamificação surgiu, porque assim como outros programas de saúde (AL-DUBAI et al., 2013), (MADEIRA et al., 2002), o nível de engajamento dos cidadãos não conseguiu se manter por muito tempo no VazaDengue. Análises baseadas em métricas, tais como número de novos usuários e número de visualizações reforçaram este entendimento, quando mostraram que o nível de engajamento dos cidadãos foi constantemente caindo ao longo do tempo. A Figura 1 mostra este cenário no período de Abril de 2015 a Abril de 2018.

Entretanto o presente estudo irá mostrar que a tarefa de se introduzir gamificação num sistema existente não é tão simples, principalmente quando existem fatores complicantes, tais como uma infraestrutura limitada, um sistema com documentação precária, pouca experiência dos envolvidos em relação a gamificação, e a dificuldade de como fazer para que a qualidade arquitetural existente não seja afetada negativamente.

Mais especificamente, para avaliar os aspectos qualitativos da arquitetura antes, durante e depois do processo de gamificação, algumas ferramentas de análise arquitetural estática foram avaliadas e utilizadas, para medir o estado do sistema em cada uma dessas

Fonte: Adaptado do artigo sobre desafios de gamificação do VazaZika a ser publicado

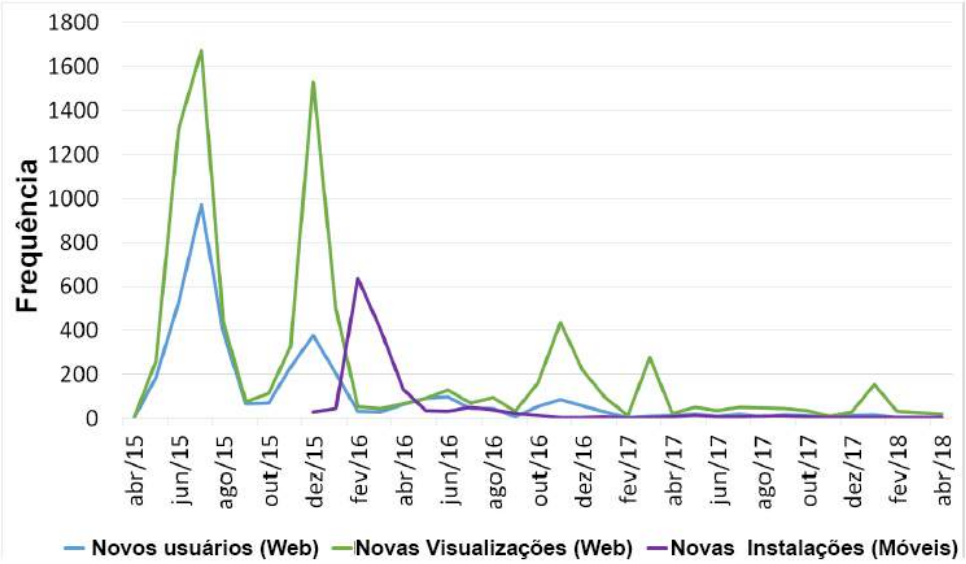

Figura 1 – Engajamento dos usuários do VazaDengue ao longo do tempo

etapas. Métricas relativas aos níveis de acoplamento e ciclidade do sistema, tais como número de dependências dos componentes, média de dependências por componente e número de componentes envolvidos em ciclos foram utilizadas para avaliar os aspectos de qualidade do sistema durante o processo de gamificação.

Os resultados obtidos a partir dos cálculos dessas métricas no sistema sugerem que a arquitetura sofreu impactos negativos durante o processo de gamificação. Esses impactos, ao serem relacionados com os desafios percebidos pelos desenvolvedores, revelaram que o processo de gamificação adotado produziu forte influência na qualidade final da arquitetura.

Este estudo fornece resultados de uma pesquisa feita sobre o processo de gamificação do então VazaDengue, possibilitando futuros pesquisadores e praticantes de gamificação se utilizarem do mesmo para retirar lições aprendidas e observar os efeitos de uma abordagem prática da inclusão da gamificação num sistema existente.

1.1 OBJETIVOS

A proposta deste trabalho é apresentada de forma macro, por meio do objetivo geral e fragmentada em tópicos definidos nos objetivos específicos.

1.1.1 Objetivo Geral

O objetivo geral desse estudo é contribuir para que acadêmicos e praticantes de gamificação tenham um exemplo disponível da introdução de elementos de gamificação num

sistema já existente e das implicações que esse processo pode causar na arquitetura de um software a depender do modo com que o processo é aplicado.

1.1.2 Objetivos Específicos

- Analisar os impactos que a gamificação causou à arquitetura do sistema;
- Comparar as percepções dos desenvolvedores, em relação aos desafios enfrentados, com o que de fato foi implementado no sistema.

1.2 METODOLOGIA

O presente estudo possui aspectos de natureza aplicada com objetivos exploratórios, que segundo Fonseca (2002), "objetiva gerar conhecimentos para aplicação prática, dirigidos à solução de problemas específicos".

Gil (2008) afirma que objetivos exploratórios têm como finalidade "desenvolver, esclarecer e modificar conceitos e ideias, com vista na formulação de problemas mais precisos ou hipóteses pesquisáveis", acarretando numa visão geral, de tipo aproximado, acerca de determinado fato e frequentemente, envolve levantamento bibliográfico e documental, entrevistas não padronizadas e estudos de caso.

Quanto à abordagem, esta pesquisa caracteriza-se, predominantemente como quantitativa, que tem raízes no pensamento positivo lógico e utiliza procedimentos estruturados e instrumentos formais para coleta de dados enfatizando a objetividade (FONSECA, 2002). Os aspectos quantitativos da pesquisa permitem que os mesmos resultados possam ser observados por diferentes pesquisadores, e normalmente são medidos em escalas numéricas, pois essas são consideradas mais ricas que descrições verbais (WAINER et al., 2007).

Além disso, essa pesquisa também se caracteriza como um estudo de caso, que segundo a abordagem de Yin (2015) trata-se de uma pesquisa empírica que investiga um evento contemporâneo dentro de seu contexto real, quando os limites entre o evento e o contexto não estão claramente definidos.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

Esta dissertação está organizada em cinco capítulos. O primeiro é responsável por realizar uma introdução ao tema da pesquisa, passando pelos objetivos geral e específicos, bem como dando uma visão geral sobre a metodologia adotada e apresentando a forma com que o trabalho está organizado.

O segundo capítulo trata dos conceitos de gamificação como uma forma de engajamento dos usuários de um sistema, passando por alguns exemplos de sucesso da aplicação dessa técnica. Neste capítulo são abordados também os conceitos de qualidade de arqui-

tetura de software, as vantagens de se prezar por um sistema desenvolvido com baixos níveis de acoplamento e altos níveis de coesão.

O capítulo 3 apresenta a metodologia do estudo de maneira detalhada, mostrando as questões e oportunidades de pesquisa, bem como as ferramentas utilizadas para executar cada passo para chegar nos resultados obtidos no estudo.

O capítulo 4 se encarrega de reunir os resultados obtidos a partir do estudo feito, analisando cada um deles e apresentando discussões a respeito dos achados da pesquisa.

No capítulo 5 são mostradas algumas conclusões obtidas a partir do estudo feito, e também as limitações do mesmo, sugerindo oportunidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados os aspectos teóricos necessários para entender o que é gamificação, discorrendo sobre alguns dos seus elementos, aspectos de motivação e alguns casos de sucesso da utilização desta técnica para promover o engajamento dos usuários nos mais diversos segmentos. Além disso, são abordados aqui os achados do estudo prévio realizado para identificar os desafios percebidos pelos desenvolvedores ao gamificarem o VazaDengue. Estudo este que serviu como ponto de partida para as análises e resultados obtidos no presente trabalho. Neste capítulo são abordados também os conceitos de arquitetura de software, relatando as características de uma arquitetura orientada a serviços e ainda ressaltando os aspectos de qualidade que devem ser observados numa arquitetura, para que esta possa fornecer condições favoráveis para sua manutenção e evolução, bem como a introdução de gamificação em suas características.

2.1 GAMIFICAÇÃO

Gamificação utiliza elementos de jogos e regras em contextos que não são jogos com o objetivo de engajar as pessoas, dando-lhes tarefas a serem executadas (DETERDING et al., 2011a). Esta técnica, segundo Kardan e Arani (2016), pode ser vista sendo aplicada com sucesso nos mais diversos segmentos, alguns deles são mostrados na figura 2, e já se mostrou como uma poderosa ferramenta para provocar uma motivação interna nas pessoas, para fazê-las continuarem interagindo com o objeto gamificado em questão e assim promovendo um maior engajamento por parte delas.

Isso é possível, porque a gamificação se encarrega de trazer alguns dos aspectos dos jogos que fazem as pessoas passarem horas concentradas, realizando várias atividades a fim de alcançarem determinados objetivos, para um contexto que não se trata de um jogo, mas sim algo que diz respeito ao mundo real. Alguns desses aspectos são discutidos na seção a seguir, como forma de elementos dos jogos que são aproveitados para promover o engajamento das pessoas.

2.1.1 Elementos de Jogos

O uso de elementos de jogos em outros contextos é um tópico antigo da intereção humano-computador (HCI) (DETERDING et al., 2011b). Tentativas de derivar heurísticas para interfaces agradáveis de jogos remontam ao início dos anos 80 (MALONE, 1980), (MALONE, 1982). Mais recentemente, pesquisadores tentaram identificar padrões de projeto que pudessem proporcionar diversão no uso sob o nome “funologia”, explicitamente inspirando-se no design de jogos (DETERDING et al., 2011b), (MONK et al., 2002).

Fonte: Adaptado de (KARDAN; ARANI, 2016)

Figura 2 – Segmentos de Gamificação

Existem diversos elementos que compõem um jogo, os quais podem ser utilizados fora desse contexto. Alguns pesquisadores criaram modelos para categorizar esses elementos, com o objetivo de aproximar os projetistas dos desenvolvedores, no sentido de fazê-los identificar de forma padronizada esses elementos e assim tornar mais fácil para todas as partes a decomposição, o estudo e a possibilidade de projetar uma ampla classe de projetos e artefatos de jogos (HUNICKE; LEBLANC; ZUBEK, 2004).

Hunicke, LeBlanc e Zubek (2004) propõem um framework chamado de MDA, o qual tem por objetivo fornecer uma abordagem formal para a compreensão de jogos. Este se divide em três camadas que dão nome ao framework: mecânicas (*mechanics*), dinâmicas (*dynamics*) e estéticas (*aesthetics*). Essas camadas representam em um jogo as regras, o sistema e a diversão, respectivamente.

- **Mecânica**: Descreve os componentes particulares de um jogo, no nível de representação dos dados e algoritmos.
- **Dinâmica**: Descreve o comportamento em tempo de execução das mecânicas agindo em consequência das entradas do jogador e cada respectiva saída no decorrer do jogo.
- **Estética**: Descreve as respostas emocionais desejadas causadas no jogador, quando este interage com o jogo.

De acordo com a Figura 3, essas camadas podem ser observadas sob duas perspectivas diferentes. A primeira diz respeito a perspectiva do *designer*, a qual tem um primeiro contato com a camada mecânica, sendo esta responsável por dar fluidez à dinâmica, levando a estética pretendida. Já na perspectiva do jogador, a estética é a porta de entrada

para entender a dinâmica e, consequentemente, a mecânica representadas no jogo (BOND, 2014).

Fonte: Adaptado de (HUNICKE; LEBLANC; ZUBEK, 2004)

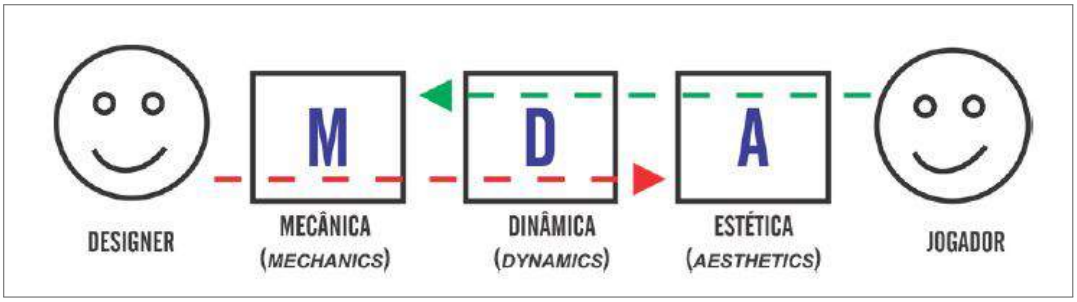

Figura 3 – Framework MDA e Suas Perspectivas

Como na abordagem de design centrado no usuário (UCD) (PEA, 1986), o design de um jogo deve dar prioridade a experiência do jogador no lugar das funcionalidades em si. Esse procedimento favorece o próprio trabalho do designer na hora de modelar os elementos do jogo (MARINS, 2013).

Dentre esses elementos que já foram utilizados na área de gamificação podemos citar: pontos, recompensas, placar de líderes, rankings, medalhas, missões, objetivos, emoção, desafios, níveis, progressão, entre outros. Cada um desses elementos pode ser entendido sob o olhar de cada uma das camadas acima mencionadas. Na prática, e na maioria dos casos, a forma como estes elementos são modelados e implementados num sistema gamificado é que vai dizer se o engajamento será um sucesso ou não. Por isso, alguns estudos são feitos para medir o sucesso dos projetos de gamificação, para analisar o comportamento dos usuários e continuamente melhorar o design da gamificação (HEILBRUNN; HERZIG; SCHILL, 2014).

O framework proposto por Hunicke, LeBlanc e Zubek (2004) foi pensado para facilitar os projetistas a idealizarem explicitamente os objetivos de design ao desenvolverem um jogo. Esta ação permite antecipar como as mudanças realizadas no projeto afetarão cada aspecto da estrutura e implementações resultantes. Além disso, os três componentes que formam o framework ajudam a conceituar o comportamento dinâmico dos jogos, e entender esse comportamento é a chave para controlar os resultados indesejados (HUNICKE; LEBLANC; ZUBEK, 2004).

Além desse modelo criado por Hunicke, LeBlanc e Zubek (2004), existem outros modelos criados por outros autores com objetivos semelhantes, é o caso de Schell (2014) que criou um modelo composto por quatro camadas, o qual chamou de tédrade elementar. Nela as categorias de elementos são descritas tendo como ponto de vista o jogador, dividindo o modelo no aspecto do que é mais e menos visível para este. Como a parte mais visível para o jogador o autor apresenta a camada estética, tendo como meio termo as

camadas mecânica e história e como menos visível a camada tecnologia, fechando assim as quatro camadas.

Outro exemplo de modelo é o de Werbach e Hunter (2012), bastante semelhante ao de Hunicke, LeBlanc e Zubek (2004), onde existem também três categorias determinadas por eles para definir os elementos de jogos, são elas: dinâmica, mecânica e componente, formando uma pirâmide na qual a categoria dinâmica está no topo sendo apoiada pela mecânica, seguida da categoria componente.

2.1.2 Motivação

Um dos aspectos mais importantes dos jogos que foi trazido para os sistemas e contextos que não são jogos, através do conceito de gamificação, diz respeito a motivação. Se de alguma forma é possível trazer os aspectos que fazem as pessoas continuarem jogando por horas e horas para um ambiente onde as atividades desempenhadas não se tratam apenas de um jogo, mas de algo que tem efeitos no mundo real, os resultados podem ser bastante promissores (BURKE, 2015).

Por isso a gamificação gira em torno disso, engajar pessoas em um nível emocional e motivá-las a alcançarem objetivos previamente estabelecidos (BURKE, 2015). Essa tarefa, no entanto, não é simples, pois para descobrir o que torna as pessoas felizes, e o que as faz continuar motivadas a jogar é necessário o entendimento de como funciona a mente humana (SCHELL, 2014).

De fato, motivação é o fator chave que a gamificação busca extrair do contexto dos jogos. É fácil encontrar exemplos de pessoas que dedicam horas do seu tempo, buscando diversão e emoções positivas através dos video games, por exemplo.

Esta motivação pode ser interna ou externa. Recebe o nome de motivação intrínsica quando a motivação é interna, ou seja, quando um indivíduo é movido por suas próprias razões, independente de um estímulo externo. Já quando o indivíduo é movido por um fator externo, ela é chamada de motivação extrínsica (ALVES, 2015).

Elementos como por exemplo medalhas e progressão de níveis são usados no contexto de gamificação com o objetivo de promover satisfação ao jogador, gerando nele um tipo de motivação intrínsica, no sentido deste buscar uma espécie de auto realização. Em alguns casos esse tipo de motivação pode ser gerada através apenas de elementos virtuais. Por outro lado, a motivação extrínseca é utilizada para motivar o jogador entregando recompensas reais, como por exemplo um cupom de desconto em determinado estabelecimento, ao atingir um determinado nível em um jogo.

Entretanto, cada pessoa ou indivíduo tem suas próprias motivações particulares que as movem, e essas motivações podem surgir de fatores internos, quando estão relacionados a razões psicológicas de cada um, ou fatores externos, quando os estímulos vêm do ambiente no qual estão inseridos (RIBEIRO et al., 2016).

Por ser a mais utilizada pelos jogos, a motivação intrínseca é a mais desejada pela gamificação, pois quando o jogador está motivado intrinsecamente, ele age em causa própria, a atividade é para ele naturalmente interessante. Estas atividades lhe atraem e geram sentimentos de satisfação e alegria (KAPP, 2012).

Sendo assim, o envolvimento do jogador torna-se espontâneo, fazendo da atividade um fim em si mesma, e desta forma, pode-se dizer que um indivíduo motivado "procura novidade, entretenimento, satisfação da curiosidade, oportunidade para exercitar novas habilidades e obter domínio" (GUIMARÃES, 2001).

Já no caso da motivação extrínseca, esta surge quando um indivíduo é motivado por recompensas vindas do ambiente externo e está relacionada ao envolvimento transacional (BURKE, 2015). Geralmente, este tipo de motivação diz respeito a algum comportamento realizado para obter uma recompensa ou evitar uma punição, e por isso, a motivação extrínseca é vista como artificial e não duradoura (KAPP, 2012).

Alguns autores alertam que é necessário cautela ao se utilizar motivadores extrínsecos, pois muitas vezes, esses motivadores podem influenciar negativamente as atitudes e comportamentos esperados. Além disso, "recompensas e punições podem provocar comportamentos não éticos e incentivar pensamentos de curto prazo, em vez de decisões de longo prazo e pensamentos reflexivos" (PINK, 2011).

Porém esse tipo de motivação não deve ser descartada, pois existem ocasiões em que os motivadores extrínsecos podem ser utilizados efetivamente. O aspecto mais importante ao se utilizar esse tipo de motivação é oferecer recompensas inesperadas e apenas após a conclusão de uma tarefa. Geralmente esse tipo de motivação deve ser utilizado quando se tratarem de tarefas que são significativas e ao mesmo tempo chatas e tediosas. E ainda assim, as recompensas devem ser utilizadas estrategicamente e esporadicamente para não gerar dependência e expectativas por parte dos jogadores (PINK, 2011).

Kapp (2012) diz ainda que devido a complexidade da natureza humana e a particularidade que cada indivíduo tem em relação a motivação, esses dois tipos de motivadores não podem ser classificados como sendo um bom e o outro ruim. A verdade é que as pessoas, na maioria das vezes, são motivadas intrínsica e extrinsicamente de maneira simultânea (KAPP, 2012). Logo, esses dois tipos de motivação não são mutuamente excludentes, podendo serem utilizados lado a lado para provocar interesse e engajamento dos jogadores (BURKE, 2015).

2.1.3 Gamificação em Contextos Relacionados a Saúde

O conceito de gamificação vem sendo usado no contexto da saúde em diversos aspectos. Alguns pesquisadores da área de saúde cogitaram que a gamificação deveria ser usada, por exemplo, para aumentar o engajamento em programas de educação médica, fazendo com que as tarefas de aprendizado se tornassem mais recompensadoras (BARGEN; ZIENTZ; HAUX, 2014).

Aplicações gamificadas tem o potencial de inspirar os pacientes a serem mais responsáveis com os seus próprios tratamentos (e.g., ajudando a melhorar a aderência às medicações) e podem ser, de maneira singular, adaptadas para engajar crianças que têm dificuldades em relatar seus sintomas (SICKKIDS, 2014), por exemplo. Tanto para adultos quanto para crianças, a esperança é que o ativo engajamento com um processo gamificado leve a um comportamento de auto-preservação, reduzida apreensão em relação aos tratamentos e um engajamento de longo prazo por parte das pessoas em relação ao profissional de saúde (GARETT; YOUNG, 2018).

Entretanto, segundo Garett e Young (2018), não existe um consenso a respeito de como a gamificação é operacionalizada e definida no contexto da saúde. Embora existam muitos estudos sendo conduzidos a respeito da gamificação sendo aplicada em programas de saúde, a concepção atual da gamificação neste domínio é muito vasta, a ponto dos pesquisadores não conseguirem replicar os achados, uma vez que não há um consenso sobre os elementos de design de aplicações de saúde gamificadas. Por essa razão, Garett e Young (2018) conduziram uma revisão sistemática, para identificar os elementos de gamificação mais popularmente usados no contexto da saúde.

De acordo com a revisão da literatura feita por Garett e Young (2018), a maioria dos estudos focam na utilização de gamificação, no contexto da saúde, para motivar o aprendizado, seja nos pacientes ou nos jovens estudantes da área. Semelhantemente, Graafland, Schraagen e Schijven (2012) conduziram uma revisão sistemática, onde puderam identificar um total de 25 artigos que descrevem a soma de 30 games sérios, os quais são usados para treinar médicos profissionais (e.g. treinamento de habilidades cirúrgicas) e agrupá-los para propósitos educacionais (PEREIRA et al., 2014).

Gill, Gill e Young (2013) afirmam que muitos estudos de educação de saúde online têm descoberto que as interações sociais são essenciais para a mudança de comportamento, com os participantes recebendo motivação e encorajamento necessários para reforçar os comportamentos através da colaboração, apoio e competição de uns com os outros.

Esses achados dão base para inferir que a gamificação ajudaria bastante na conscientização dos cidadãos a auxiliarem os profissionais de saúde a prevenirem doenças. A interação com o problema torna os jogadores participantes da solução. Essa transferência de responsabilidade sendo feita de maneira agradável, promovendo nos envolvidos o prazer em ajudar, só faz aumentar as chances de sucesso.

Thorsteinsen, Vittersø e Svendsen (2014) conduziram um estudo que testaria a efetividade de uma intervenção online de atividades físicas praticadas por um grupo de pessoas, a qual se utilizaria de componentes de jogos. No estudo, eles puderam constatar que os aspectos de jogo utilizados na intervenção foram incentivo para que o grupo praticasse mais atividades físicas durante o período de análise.

Um outro exemplo de intervenção, utilizando elementos de jogos em relação a saúde, é dado através do estudo realizado por Jones, Madden e Wengreen (2014), no qual o

consumo de frutas e vegetais é incentivado numa escola. Os resultados do estudo mostram que durante o período de intervenção, o consumo desses alimentos aumentou em mais de 30%.

Estes e outros exemplos mostram que a gamificação tem influência positiva na mudança de comportamento de seu público alvo, a fim de que este realize determinadas atividades pretendidas pelos projetistas e desenvolvedores de gamificação, no tocante a tornar as pessoas engajadas num propósito específico.

2.2 GAMIFICAÇÃO DO VAZADENGUE

O VazaDengue, como já foi brevemente apresentado na introdução deste trabalho, é uma plataforma de software criada em 2015, como fruto da parceria entre a UFAL (Universidade Federal de Alagoas), PUC-Rio (Pontifícia Universidade Católica do Rio deJaneiro) e Newcastle University, com o intuito de auxiliar os agentes de saúde na identificação e eliminação de pontos de focos do mosquito da Dengue, Chikungunya e Zika.

A plataforma disponibilizava aos cidadãos a oportunidade de denunciar estes pontos de forma prática, através de uma aplicação móvel, a qual podia ser instalada em seus dispositivos Android ou iOS, ou ainda através de uma aplicação Web, podendo ser acessada nos navegadores mais populares. Este acesso facilitado permitia aos cidadãos se tornarem coparticipantes do trabalho dos agentes de saúde, informando-lhes, com mais agilidade, os locais com necessidade de atendimento mais urgente.

Como já mencionado, a adesão e engajamento dos cidadãos à plataforma VazaDengue foi diminuindo com o passar do tempo. Este fator preocupou os envolvidos no projeto, fazendo-os pensar em alguma solução que pudesse evitar ou, pelo menos, amenizar a falta de interesse das pessoas em continuar interagindo e fornecendo dados à plataforma, colaborando assim com os agentes de saúde.

Ao investigarem sobre meios de melhorar o engajamento das pessoas com os sistemas, o tema de gamificação veio à tona. A partir daí, o projeto de gamificar a plataforma tornou-se prioridade. Porém, os primeiros desafios já se apresentaram logo no início do projeto. Rapidamente surgiram perguntas como: qual abordagem utilizar? Que elementos usar? A plataforma atual tem condições de dar suporte a este processo?

Diante desses desafios, foram formados então os times que seriam responsáveis por planejar, conduzir e implementar o processo de gamificação na plataforma. Um time específico foi designado para estudar os conceitos de gamificação, a fim de oferecer as informações necessárias para a tomada de decisão quanto a qual abordagem utilizar na plataforma. Um outro time foi designado para lidar com a estrutura de *backend* da plataforma, para identificar oportunidades de reuso e fornecer informações a respeito de como essa parte do sistema poderia ser adaptada às necessidades de gamificação. Ainda outros dois times foram formados para lidar com a parte de *frontend* da plataforma, um para atender às necessidades da aplicação móvel e outro para atender a aplicação Web. Ao todo foram

formados quatro times para atuar no processo de gamificação da plataforma, sem contar com os responsáveis pelo gerenciamento das atividades dos times.

2.2.1 Time de Gamificação

As atividades do time de gamificação foram levadas em maior consideração, por se tratar do ponto focal deste trabalho, no tocante à comparação entre os desafios percebidos pelos desenvolvedores durante a gamificação de um sistema existente e os impactos deste processo na arquitetura do sistema. Os outros times também participaram do estudo feito anteriormente, justamente para ajudar os pesquisadores a entenderem e categorizarem os desafios de gamificar um sistema existente.

Os primeiros passos dados pelo time de gamificação foram em direção ao estudo dos conceitos e verificação de casos de sucesso, tanto na literatura, como na indústria. No caso da literatura, nomes como os de Deterding, Werbach e Hunter logo surgiram como importantes pesquisadores da área de gamificação. Na indústria, casos como os do *Foursquare* [1], da *Nike* [2] e do *Waze* [3] rapidamente surgiram como exemplos de sucesso de gamificação.

Após os estudos, a equipe de gamificação percebeu que o modelo proposto por Werbach e Hunter (2012) poderia ser usado no VazaZika, por se tratar de um estudo bem detalhado, podendo ser utilizado como guia para adaptar os conceitos de gamificação às necessidades da nova plataforma. Elementos da categoria *componentes* do modelo, como *badges* ou medalhas, *ranking*, pontos, missões, avatares, foram observados como possíveis integrantes do jogo na plataforma.

O próximo passo seria definir os *mecanismos* adequados à plataforma. Como o objetivo principal da plataforma é estimular a participação dos cidadãos a denunciarem pontos de focos do mosquito transmissor das doenças, os elementos de jogo, tais como desafios, *feedback* e recompensas foram cotados a participar do jogo a ser implementado na plataforma, com o intuito de promover o engajamento das pessoas.

Por fim, na categoria *dinâmicas* os elementos progressão e restrições foram escolhidos para atribuir coerência e padrões regulares a experiência que seria adquirida pelos jogadores. Estes elementos promoveriam o comportamento dinâmico do jogo, dando, de certa forma, uma sensação de evolução ao jogador ao interagir com o sistema repetidas vezes.

Durante e após esses estudos e definições, várias reuniões com os outros times foram realizadas, com o intuito de validar, descobrir a viabilidade da implantação dos conceitos, diante da realidade da plataforma e infraestrutura existentes e criar protótipos que pudessem nortear a futura implementação da gamificação na plataforma. O desafio agora era saber quais elementos poderiam ser explorados, quais deles seriam priorizados e quais deveriam ser postergados ou descartados. Com essas definições concluídas, a implemen-

[1] Empresa de tecnologia - pt.foursquare.com
[2] Empresa de artigos esportivos - about.nike.com
[3] Serviço de GPS - waze.com

tação poderia então ser iniciada. Vale ressaltar, que mesmo na fase de implementação, vários pontos discutidos anteriormente precisaram ser reavaliados, para dar seguimento ao processo de gamificação, diante das limitações da plataforma existente. Estas limitações são mais detalhadas no Capítulo 4, em termos de comparação entre o que foi percebido como desafiador pelos desenvolvedores e como as decisões tomadas ao fazer o design do novo sistema afetou a arquitetura existente.

2.2.2 Desafios da Gamificação

Para descobrir os desafios enfrentados pelos desenvolvedores durante a gamificação, um estudo liderado por pesquisadores da PUC-Rio foi conduzido a partir da elaboração de 3 perguntas de pesquisa, são elas:

- **QP1** - Quais foram as atividades de desenvolvimento percebidas como desafiadoras pelos desenvolvedores ao longo da gamificação do sistema?
- **QP2** - Por que certas atividades foram percebidas como desafiadoras ao longo da gamificação do sistema?
- **QP3** - Quais soluções os desenvolvedores deram para tentar resolver esses desafios?

2.2.2.1 Protocolo de Elaboração das Entrevistas

Para responder as questões de pesquisa, os pesquisadores realizaram entrevistas com os desenvolvedores de cada time de desenvolvimento do VazaZika. Foi escolhido o formato de entrevista semi-estruturada (RUNESON; HÖST, 2009) com o objetivo de alcançar uma maior flexibilidade durante as entrevistas. A seguir são mostrados os três passos conduzidos para a elaboração desta fase do estudo.

Passo 1: Definir os artefatos da entrevista. Foram definidos e revisados os seguintes artefatos: *Formulário de Caracterização do Participante*, o qual foi responsável por coletar as experiências passadas dos participantes, para identificar os níveis de conhecimento em algumas atividades de engenharia de software, cujos resultados são mostrados na Figura 4.

O *Roteiro da Entrevista*, o qual segue uma estrutura de funil (RUNESON; HÖST, 2009) que começa com perguntas mais genéricas e termina com questões mais específicas. O Roteiro é composto por nove questões no total, sendo duas direcionadas a confirmação do time ou times de desenvolvimento no qual o desenvolvedor esteve envolvido; uma questão para elicitar todas atividades percebidas como desafiadoras; quatro questões para entender cada atividade; e duas questões para descobrir as soluções direcionadas a cada atividade mencionada.

Passo 2: Aplicar o formulário de caracterização do participante. Foram convidados ao todo 17 desenvolvedores dos quatro times envolvidos no projeto para preencher o

formulário de caracterização. Dois participantes foram descartados, por terem colaborado muito pouco em seus respectivos times. Logo, um total de 15 desenvolvedores participaram das entrevistas.

Passo 3: Conduzir as entrevistas. As entrevistas foram conduzidas com os 15 desenvolvedores do VazaZika. O roteiro foi estritamente seguido de forma a evitar qualquer tipo de influência nas respostas dos entrevistados. As entrevistas face a face em laboratórios foram priorizadas sempre que possível, para evitar qualquer desvio de atenção ou desconcentração. Durante as entrevistas foram tomadas notas das respostas dos entrevistados e além disso foi pedida a permissão destes para que os áudios das entrevistas pudessem ser gravados. Cada entrevista durou em média uma hora e vinte e cinco minutos.

2.2.2.2 Protocolo de Análise dos dados

Após entrevistar cada participante, o próximo passo consistiu em analisar os dados das entrevistas. Devido a natureza qualitativa do estudo, foram aplicadas algumas fases dos conhecidos procedimentos de Grounded Theory (GT) (CORBIN; STRAUSS et al., 2008), (EASTERBROOK et al., 2008) para analisar as respostas dos entrevistados. Estes procedimentos foram úteis tanto para elicitar como para entender cada dificuldade enfrentada pelos desenvolvedores enquanto estes gamificavam a plataforma. A seguir são detalhados os quatro passos seguidos para fazer a análise dos dados.

Passo 1: Tabular as notas das entrevistas. Todas as respostas dos entrevistados foram colocadas em uma planilha. Logo após, os envolvidos no estudo validaram os resultados da tabulação.

Passo 2: Realizar a codificação das notas das entrevistas. Foram aplicadas duas fases de análise de dados da GT (EASTERBROOK et al., 2008) a seguir. *Open coding*, com o objetivo de elicitar os seguintes itens: atividades desafiadoras, fatores de influência e soluções aplicadas. Todos os dados foram identificados a partir dos dados das entrevistas que foram armazenados na planilha. Como exemplo de atividade desafiadora nós temos: escolher os elementos de gamificação a serem implementados a fim de engajar os usuários.

A outra fase GT aplicada foi a criação de rótulos para cada item, ou seja, cada atividade desafiadora recebeu um identificador. Como exemplo de código nós temos: [GRP03c] Elicitar os elementos de jogo para engajar os usuários. Um código axial foi aplicado para rotular os grupos de atividades desafiadoras com categorias, tais como [GRP03] Especificar o Sistema Gamificado. Dessa forma, as atividades podiam ser reunidas em uma categoria específica, bem como os grupos dessas atividades podiam ser analisados sob o ponto de vista de dois possíveis tipos de relacionamento: entre duas categorias (grupo-grupo) e entre fator de influência e um grupo (fator-grupo).

Passo 3: Analisar os áudios das entrevistas. Os áudios das entrevistas foram ouvidos a fim de validar as notas que foram tomadas durante as entrevistas, corrigir possíveis inconsistências e extrair citações dos entrevistados, o que ajudou a justificar

cada fator de influência que poderia ter gerado uma atividade desafiadora. Para dar maior credibilidade a análise, as gravações das entrevistas não foram ouvidas pelos mesmos que as conduziram.

Passo 4: Construir um mapa de atividades desafiadoras. Itens redundantes foram descartados, os rótulos das categorias e códigos foram refinados, e foram validados os dois tipos de relacionamento mencionado no Passo 2. Com isto, foi possível criar uma tabela com todos os grupos de atividades desafiadoras. Como resultado, um mapa de grupos de atividades desafiadoras foi criado, o qual representa: cada grupo com o respectivo rótulo baseado na Tabela 1; os dois tipos de relacionamento; e os fatores que influenciaram a percepção dos desenvolvedores a respeito das atividades desafiadoras.

2.2.2.3 Experiência dos Participantes

A partir do *Formulário de Caracterização do Participante*, foi observado que os desenvolvedores tinham diferentes experiências. A Figura 4 resume a experiência dos 15 desenvolvedores numerados de D1 a D15. Suas habilidades foram verificadas em relação a oito conceitos da engenharia de software, ex: gamificação. Os resultados sugerem que os desenvolvedores tinham de alto a muito alto conhecimento em conceitos básicos, tais como testes de software (41%), engenharia de requisitos (59%), e desenvolvimento web (65%). Por outro lado, os desenvolvedores tinham de baixo, muito baixo a nenhum conhecimento em gamificação (82%) e game design (89%). A média de experiência em desenvolvimento dos participantes é de seis anos.

Fonte: Adaptado do artigo sobre desafios de gamificação do VazaZika a ser publicado

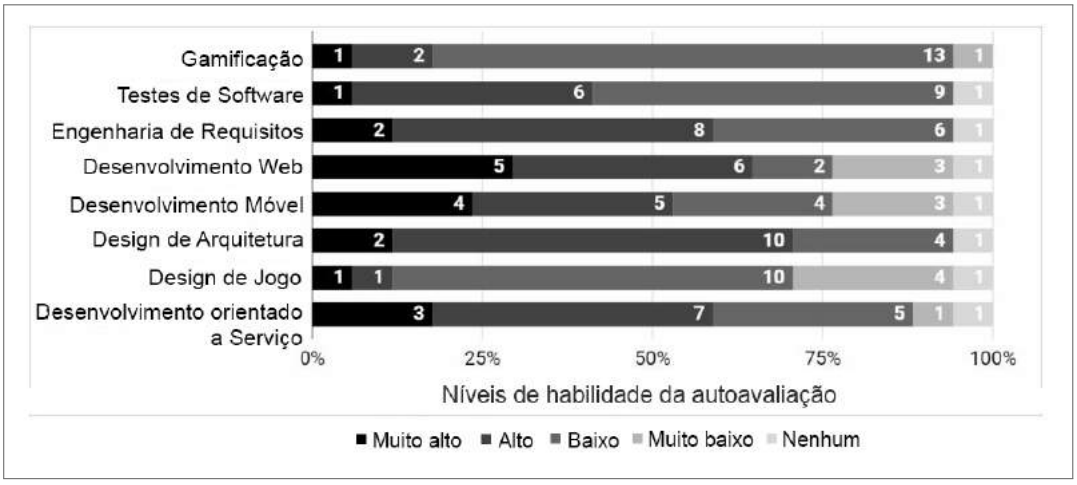

Figura 4 – Auto-Avaliação das Habilidades dos Desenvolvedores

2.2.2.4 Atividades Desafiadoras de Gamificação

A Tabela 1 apresenta os grupos de atividades desafiadoras elicitados a partir das entrevistas com os desenvolvedores. Cada grupo dispõe de uma descrição, uma lista de

atividades desafiadoras que compõem o grupo e em que período cada atividade surgiu durante a gamificação do VazaDengue. Por exemplo, [GRP01] é um grupo de atividades desafiadoras relacionado a tomada de decisão a respeito do sistema gamificado. [GRP01] é composto de atividades desafiadoras, como por exemplo [GRP01a] Definir o sistema VazaZika.

Tabela 1 – Grupos de Atividades Desafiadoras

[GRP01] Definir o Sistema Gamificado	**[GRP02] Conceber a Arte & Design do Sistema**	**[GRP03] Especificar o Sistema Gamificado**
Descrição: Relacionado à tomada de decisão sobre o sistema gamificado, em termos de organização do código fonte, decomposição dos serviços e persistência dos dados. **Atividades Desafiadoras:** [GRP01a] Definir o sistema VazaZika; [GRP01b] Modelar a arquitetura Web do VazaZika; [GRP01c] Modelar o banco de dados; **Surgimento:** (a) a partir do início do projeto e ao longo de sua execução; (b) ao longo da implementação da gamificação; (c) após a primeira especificação do sistema gamificado.	**Descrição:** Relacionado ao processo criativo de aspectos de arte & design (formas visuais, cores, entre outros) do sistema gamificado. **Atividades Desafiadoras:** [GRP02a] Conceber a arte & design do sistema; **Surgimento:** (a) a partir do início do projeto, especialmente após a proposição do modelo de gamificação.	**Descrição:** Relacionado à definição dos elementos e regras do jogo para engajar os usuários através da gamificação. **Atividades Desafiadoras:** [GRP03a] Especificar o sistema gamificado; [GRP03b] Garantir a conformidade do design de gamificação; [GRP03c] Elicitar os elementos do jogo que objetivavam engajar os usuários; [GRP03d] Número de envolvidos insuficiente para prover feedback; [GRP03e] Integrar as redes sociais com o sistema gamificado. **Surgimento:** (a) a partir do início do projeto, especialmente durante a implementação do sistema; (b) a partir do início do projeto; (c) ao longo da implementação do sistema.
[GRP04] Implementar o Sistema Gamificado	**[GRP05] Entender o Sistema Legado**	**[GRP06] Gerenciar os Times de Desenvolvimento**
Descrição: Relacionado à implementação do sistema gamificado. **Atividades Desafiadoras:** [GRP04a] Desenvolvimento iOS; [GRP04b] Desenvolvimento mobile; [GRP04c] Inexperiência com desenvolvimento mobile híbrido; [GRP04d] Entedimento do domínio de gamificação; [GRP04e] Deploy manual do sistema; [GRP04f] Sincronizar os dados dos usuários do sistema. **Surgimento:** (a) a partir do início do projeto; (b) a partir do início do desenvolvimento mobile; (c) ao longo da implementação do sistema.	**Descrição:** Relacionado às dificuldades enfrentadas para entender o sitema VazaDengue, em termos de contexto, design de arquitetura, tecnlogias e código fonte. **Atividades Desafiadoras:** [GRP05a] Entender a API do VazaDengue; [GRP05b] Entender a camada de backend; [GRP05c] Entender o domínio do sistema. **Surgimento:** (a) ao longo da implementação do sistema; (b) após um especialista no desenvolvimento do VazaDengue deixar o projeto; (c) a partir do início da implementação do sistema.	**Descrição:** Relacionado ao gerenciamento das tarefas dos times de desenvolvimento. **Atividades Desafiadoras:** [GRP06a] Atribuir tarefas aos membros dos times; **Surgimento:** (a) a partir do início do projeto e a partir dos primeiros atrasos das tarefas.
[GRP07] Comunicação entre os Times de Desenvolvimento		**[GRP08] Testar o Sistema Gamificado**
Descrição: Relacionado à comunicação entre os membros de diferentes times de desenvolvimento. **Atividades Desafiadoras:** [GRP07a] Comunicar os membros dos times; [GRP07b] Especificação do sistema gamficado negligenciada; **Surgimento:** (a) a partir do início do projeto e a partir da definição dos membros por time; (b) a partir das primeiras entregas da impelementação do sistema.		**Descrição:** Relacionado à escrita e execução dos testes unitários. **Atividades Desafiadoras:** [GRP08a] Escrever casos de teste; **Surgimento:** (a) durante mudanças na fase de implementação de funcionalidades existentes.

Fonte: Adaptado do artigo sobre desafios de gamificação do VazaZika a ser publicado

A partir dessa tabela os desafios foram extraídos para serem futuramente comparados aos impactos causados na arquitetura do sistema.

Por fim, foram reunidas as soluções relatadas pelos desenvolvedores nas entrevistas, de forma a agrupá-las de acordo com quais desafios elas pretendiam resolver ou contornar. A Tabela 2 apresenta as soluções endereçadas a cada atividade desafiadora, descrevendo seu propósito.

Tabela 2 – Soluções Endereçadas às Atividades Desafiadoras

ID	Solução	Propósito	Atividades Desafiadoras
[SOL01]	Engenharia Reversa	Entender sistema legado: código fonte, banco de dados e design	[GRP01a, GRP04d, GRP05c, GRP05a, GRP05b]
[SOL02]	Geração automática de documentação de API	Evitar documentação de software precária	[GRP05a]
[SOL03]	Inspeção de conformidade com o design de gamificação	Alinhar o que foi implementado com o que foi especificado	[GRP02a, GRP07b, GRP07a, GRP03b, GRP04d]
[SOL04]	Refinamento incremental do design de gamificação	Melhorar a atratividade do sistema e engajamento dos usuários	[GRP07a, GRP07b, GRP03c]
[SOL05]	Inspeção de protótipo em pares	Alinhar o design especificado com os protótipos	[GRP04]
[SOL06]	Wireframe	Desenhar ambas interface e navegação do sistema	[GRP02a]
[SOL07]	Persona	Elicitar requisitos funcionais a partir do ponto de vista de um usuário candidato	[GRP02a, GRP03c, GRP03d]
[SOL08]	Constantes encontros entre os envolvidos no projeto	Alinhar o design de gamificação com as necessidades dos agentes de saúde e dos cidadãos	[GRP03d]
[SOL09]	Framework de aplicação mobile híbrida	Desacoplar camadas de back e front-end para promover o reuso e a portabilidade	[GRP01a, GRP04b]
[SOL10]	Gerenciador automático de deploy do sistema	Reduzir a complexidade de gerenciamento das dependências da aplicação web	[GRP04e]

Fonte: Adaptado do artigo sobre desafios de gamificação do VazaZika a ser publicado

2.3 ARQUITETURA DE SOFTWARE

Segundo Soni, Nord e Hofmeister (1995), a arquitetura de software descreve como um sistema é decomposto em componentes, como estes componentes estão interconectados e como eles se comunicam e interagem entre si. Quando mal interpretados, estes aspectos de design são a maior fonte de erros. Hofmeister, Nord e Soni (2000) afirmam ainda que um fator crítico para se ter um produto de software de sucesso depende de uma boa arquitetura de software entendida pelos interessados no projeto (em seu nível apropriado de detalhes) e pelos membros do time de desenvolvimento.

Hofmeister, Nord e Soni (2000) dizem também que embora uma boa arquitetura não garanta que um produto de software represente de fato os seus requisitos, uma arquitetura pobremente desenhada e mal definida torna essa tarefa quase impossível. A medida que os componentes de um software são arranjados e implementados de maneira irresponsável, em relação a qualidade arquitetural, mais dificilmente ele atenderá aos propósitos e objetivos para os quais foi projetado.

As preocupações em relação a qualidade arquitetural não se restringem somente ao momento da modelagem inicial. Estudos mostram que a deterioração dos sistemas de software ao longo do tempo vem sendo discutida desde os anos 60, como um debate sobre as “crises de software” (SILVA; BALASUBRAMANIAM, 2012). A necessidade da evolução dos softwares, para atender novos requisitos e fornecer melhorias, faz com que este fique suscetível à perda da qualidade de sua arquitetura, por causa da implementação de novas funcionalidades e ajustes realizados no código existente.

A este fenômeno de perda de qualidade arquitetural dá-se o nome de erosão arquitetural. Ele representa o sinal de que a qualidade arquitetrual inicial do software foi reduzida (JAKTMAN; LEANEY; LIU, 1999). Esta redução é medida a partir de diversos fatores, porém um dos fatores mais importantes diz respeito a capacidade que o software tem de acomodar mudanças implementadas (JAKTMAN; LEANEY; LIU, 1999). Esta capacidade deve ser constamente avaliada, pois um software com arquitetura erodida se torna mais suscetível a erros, incorre em altos custos de manutenção, compromete seu desempenho e, obviamente, é levado a erodir ainda mais sua arquitetura (SILVA; BALASUBRAMANIAM, 2012).

A despeito dessa realidade, o processo para resolver esse tipo de problema nos softwares é feito de maneira *ad hoc*, sem o suporte adequado de uma ferramenta de nível arquitetural. Por esta razão, alguns estudos já apresentam algumas soluções em forma de guias para desenvolvedores e mantenedores de software, no sentido de ajudarem estes no trabalho de reverter o eventual processo de erosão arquitetural que venha existir em seus sistemas (CZARNECKI et al., 2012).

Uma das formas de previnir a erosão arquitetural se dá através do uso de boas ferramentas e técnicas que auxiliem os arquitetos de software a estabelecerem regras e restrições previamente, a fim de dar ciência aos desenvolvedores, quando estes estiverem prestes a cometer algum tipo de erro arquitetural. Passos et al. (2010) dão uma visão geral de três técnicas que engenheiros e arquitetos podem utilizar para ajudá-los a analisarem a conformidade arquitetural de um software, isto é, checar se a arquitetura do sistema de software implementado é consistente com a visão de arquitetura do módulo previamente definida (KNODEL; POPESCU, 2007).

Esta visão define a organização estática de um sistema em elementos estruturais (tais como pacotes, subsistemas, e camadas) e como tais elementos devem interagir entre si (KRUCHTEN, 1995). Normalmente, arquitetos usam esta visão para planejar e alocar seus times de trabalho, para avaliar o progresso da implementação, para racionalizar a respeito das oportunidades de reuso do software e para estabelecer linhas de produto de software (PASSOS et al., 2010).

A seguir, os conceitos e aspectos de qualidade de arquitetura de software serão mais detalhadamente abordados. Para uma melhor compreensão dos elementos que podem melhorar ou piorar a qualidade arquitetural, forneceremos uma visão geral dos fatores que podem contribuir para uma possível erosão arquitetural.

2.3.1 Aspectos de Qualidade

Quando se fala em qualidade de software, muitos termos como usabilidade, testabilidade, manutenibilidade, confiabilidade, segurança, disponibilidade, performance, entre outros, surgem tanto na literatura, quanto na indústria. De acordo com a ISO 9126-1 (ISO/IEC, 1998), qualidade é definida como um conjunto de recursos e características de

um produto ou serviço que se apoia em sua habilidade de satisfazer necessidades declaradas ou implícitas (LOSAVIO et al., 2003).

Trazendo para um contexto mais específico, um software pode ter sua qualidade analisada sob o ponto de vista de sua arquitetura. Como já mencionado por Soni, Nord e Hofmeister (1995), esta arquitetura é responsável por descrever a decomposião e relacionamento dos componentes de um sistema. Sendo assim, é possível inferir que o cuidado na organização e disposição desses componentes pode contribuir para uma arquitetura com boa qualidade.

Segundo Dhama (1995), as duas propriedades de software que tem um grande impacto na qualidade do software são *cohesion* e *coupling* (C & C). A Figura 5 mostra que 8 dos 13 fatores, identificados por Rome Laboratory (BOWEN; POST; TSAI, 1983), são dependentes dessas duas propriedades. Sendo assim, a identificação, medição e gerenciamento do C & C nos sistemas de software podem ter uma maior influência na redução dos custos deste.

Fonte: Adaptado de (BOWEN; POST; TSAI, 1983)

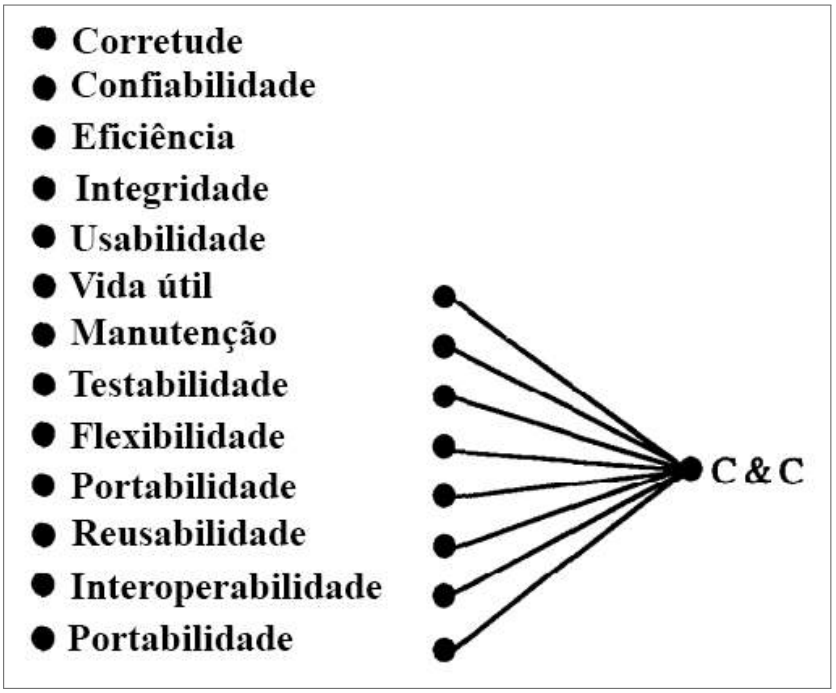

Figura 5 – Hierarquia de qualidades de software propostas por Rome Laboratory (BOWEN; POST; TSAI, 1983)

Ainda de acordo com Dhama (1995), a *Coesão* em um módulo refere-se àquela propriedade de software que liga as várias declarações feitas no código e os outros módulos menores contidos neste. Ele define um módulo como sendo uma unidade de código compilável e diz que a *Coesão* é uma propriedade intramódulo que reflete as considerações de design para integrar os vários componentes do módulo em uma única unidade. A força e consequentemente a qualidade do módulo aumenta, à medida que a *Coesão* também aumenta. Dhama (1995) explica ainda que *Acoplamento* é uma medida da interdependên-

cia entre dois módulos de software. É uma propriedade intramódulo e é desejável que as mudanças feitas num módulo afetem o mínimo possível um outro módulo. Portanto, a qualidade de um módulo aumenta, à medida que o *Acoplamento* diminui.

Apesar dessas duas propriedades terem sido analisadas em duas frentes diferentes no estudo acima mencionado, elas estão intimamente ligadas. Pois quando um módulo ou componente se associa a outro desnecessariamente, cria-se então um nível de acoplamento entre eles, provocando assim uma perda de coesão no sistema, por se tratar de um relacionamento que conceitualmente não deveria existir.

Geralmente, na prática, este tipo de relacionamento desnecessário não é criado intencionalmente, principalmente quando a implementação da arquitetura de software é feita por profissionais experientes. No entanto, mesmo os engenheiros de software mais experientes podem acabar deixando passar despercebido uma propriedade que causa acoplamento entre módulos e componentes. Esta propriedade é conhecida como *Ciclicidade*.

2.3.1.1 Acoplamento e Ciclicidade

Um aspecto importante na avaliação dos níveis de acoplamento de uma arquitetura está ligado à quantidade de componentes envolvidos em ciclos. A Figura 6 mostra um paralelo entre duas formas distintas de organizar a arquitetura de um sistema. À esquerda é possível perceber que os componentes estão dispostos em uma estrutura cíclica, enquanto na direita, esses mesmos componentes estão organizados em uma estrutura nivelada. No primeiro caso, uma mudança feita em qualquer dos componentes representa grandes chances de afetar os demais. Já no segundo caso, uma mudança feita no componente C, por exemplo, não afetaria os outros componentes.

Logo, é possível chegar ao consenso de que um sistema com menos estruturas cíclicas e com baixos níveis de acoplamento possui uma qualidade arquitetural melhor que a de um sistema que não apresenta essas características.

Para colocar esse consenso em termos práticos, podemos fazer alguns cálculos simples com os dois tipos de arquitetura acima mencionados, utilizando os valores das métricas referentes ao número de dependências de cada componente (DependsOn), o número cumulativo de dependências (Cumulative Component Dependencies - CCD) e a média de dependências por componente (Average Component Dependencies - ACD).

Para o primeiro caso temos:

- **DependsOn:** 3 para cada componente. Pois todos dependem de si mesmos e dos demais.
- **CCD:** $3 + 3 + 3 = 9$.
- **ACD:** $9/3 = 3$.

Para o segundo caso temos:

Fonte: Adaptado do blog da ferramenta SonarGraph

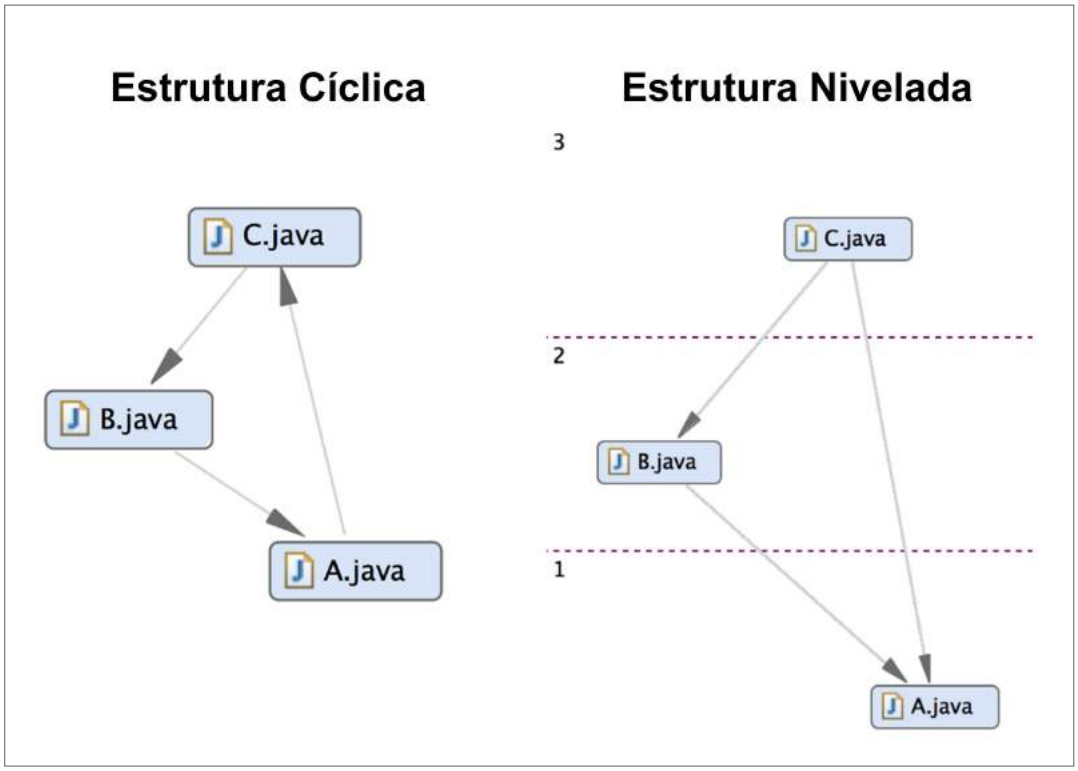

Figura 6 – Modelos de Arquitetura de Software

- **DependsOn:** 1 (A), 2 (B), 3 (C).
- **CCD:** $1 + 2 + 3 = 6$.
- **ACD:** $6/3 = 2$.

Segundo (LAKOS, 1996), dependências físicas cíclicas entre componentes inibem a compreensão, o teste e a reutilização. Portanto, para avaliar a qualidade arquitetural de um sistema, este precisa também ser avaliado no sentido de descobrir se há esse tipo de estrutura em sua arquitetura. Utilizando ainda o exemplo da Figura 6, no primeiro caso temos o número de componentes envolvidos em ciclos igual a três, enquanto que no segundo caso nenhum dos componentes está envolvido em estrutura cíclica.

2.4 CONTROLANDO A EROSÃO ARQUITETURAL

Alguns estudos foram realizados para descobrir formas de lidar com a questão da erosão arquitetural nos sistemas. (SILVA; BALASUBRAMANIAM, 2012) é o exemplo de um desses estudos. Eles classificaram abordagens existentes para controlar a erosão arquitetural em três categorias dependendo de se elas tentam *minimizar*, *prevenir* ou *reparar* a erosão. Cada uma dessas categorias contém uma ou mais sub-categorias baseadas nas estratégias de alto nível usadas para descobrir seu objetivo.

Resumidamente, a categoria *minimizar* contém aquelas estratégias relacionadas a limitar a ocorrência e impacto da erosão arquitetural, mas que podem não ser efetivas em eliminá-la. Por outro lado, as estratégias presentes na catetoria *prevenir* têm por objetivo erradicar completamente a erosão arquitetural. Em terceiro lugar está o conjunto de estratégias que tentam reparar os danos causados pela erosão arquitetural e tentar reconciliar a implementação com sua arquitetura (SILVA; BALASUBRAMANIAM, 2012). Algumas dessas estratégias são mais detalhadas abaixo.

A) *Minimizar*

- **Processo orientado a conformidade da arquitetura**: inclui processos de engenharia de software que certificam a conformidade da arquitetura durante as atividade de desenvolvimento e manutenção dos sistemas;
- **Gerenciamento de evolução da arquitetura**: cobre métodos disponíveis para gerenciar a evolução das especificações arquiteturais em paralelo com os artefatos de implmentação;
- **Aplicação de design de arquitetura**: incorpora os métodos e ferramentas disponíveis para transofrmar modelos arquiteturais em implementação.

B) *Prevenir*

- **Ligação da implementação com a arquitetura**: inclui mecanismos que associam ou incorporam modelos arquiteturais no código fonte e dá suporte a habilidade de monitorar a conformidade arquitetural em tempo de execução;
- **Tecnologias de autoadaptação**: habilitam os sistemas a reconfigurarem-se para se alinhar a sua própria arquitetura após uma mudança ter sido realizada para ambos estados de implementação e tempo de execução.

C) *Reparar*

- **Recuperação de arquitetura**: envolve extrair a arquitetura implementada e outros artefatos do código fonte;
- **Técnicas de descoberta de arquitetura**: são úteis para elicitar a arquitetura pretendida a partir das propriedades do sistema emergente e através de outros meios na falta de uma documentação arquitetural;
- **Métodos de reconciliação com a arquitetura**: ajudam a reduzir a distância entre a implementação e a arquitetura pretendida do sistema de software.

(SILVA; BALASUBRAMANIAM, 2012)

Para o contexto do presente estudo, a categoria *reparar* recebeu maior atenção, uma vez que as categorias *minimizar* e *prevenir* não se aplicam ao contexto do estudo, porque o

sistema não se encontra mais em fase de implementação. A ideia agora é tentar resgatar ao máximo a proposta da arquitetura original, reparando os erros arquiteturais para diminuir a erosão arquitetural possivelmente causada pelo processo de gamificação.

3 METODOLOGIA

Neste capítulo é apresentada a metodologia do estudo de caso do VazaZika, contextualizando sobre o estado do sistema antes de ser gamificado, mostrando os motivos que levaram às necessidades de gamificação, bem como apresentando a arquitetura antes e depois desse processo. Na ocasião, são exploradas as oportunidades de pesquisa identificadas e o detalhamento da abordagem de cada oportunidade, discorrendo sobre todos os processos seguidos para obter os resultados e conclusões posteriormente discutidos.

3.1 ARQUITETURA

A princípio, a plataforma VazaDengue contava com um servidor Tomcat, o qual era responsável por hospedar a API do VazaDengue, respondendo as requisições que vinham por meio dos dispositivos Android, iOS e da aplicação Web. Nesta API estavam implementadas as regras de negócio do sistema, para permitir as denúncias de pontos de focos do mosquito, bem como casos de doenças reportados pelos cidadãos, e ainda dar condições aos agentes de saúde de verificar todas essas denúncias.

Na Figura 7 é possível identificar o servidor Tomcat mencionado acima, além de outros servidores, tais como um servidor responsável por fazer o rastreamento de possíveis denúncias por meio das redes sociais Instagram e Twitter, um servidor responsável por fazer a classificação dos dados que vinham dessas redes sociais, um servidor apache, responsável por fazer a intermediação entre o Tomcat e os clientes da API, além de um banco de dados.

Fonte: Adaptado da apresentação do workshop do VazaZika

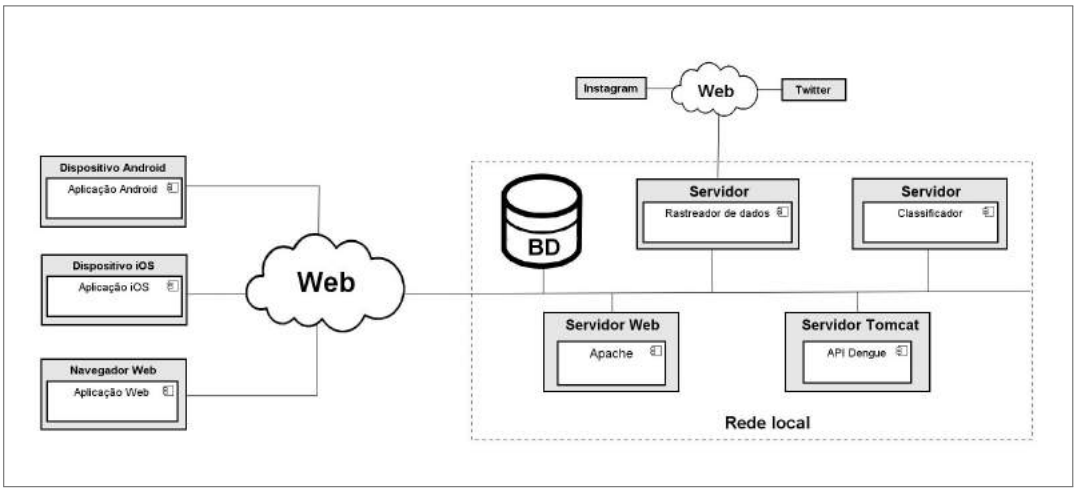

Figura 7 – Arquitetura do VazaDengue

Com a necessidade de implementação da gamificação, esta arquitetura precisou ser

reformulada em alguns pontos, entre eles podemos destacar que a API do VazaDengue foi dividida em duas partes: um componente núcleo, o qual abriga as camadas de serviços e repositórios, e um componente API, o qual comporta todos os controladores que respondem as requisições do módulo cliente.

Foi necessário ainda a introdução de uma API específica, responsável por atender as requisições relacionadas a parte de gamificação do sistema, além da adição de um componente de gamificação no núcleo, a fim de computar os elementos do jogo. A camada de persistência, por sua vez, sofreu diversas alterações para atender aos requisitos de gamificação. Por fim, foi criado um módulo cliente, o qual compreendia a aplicação Web e uma aplicação móvel. Esta última foi desenvolvida com uma tecnologia híbrida, de forma a diminuir o esforço de implementação, para disponibilizar o sistema gamificado nas plataformas Android e iOS.

Fonte: Adaptado da apresentação do workshop do VazaZika

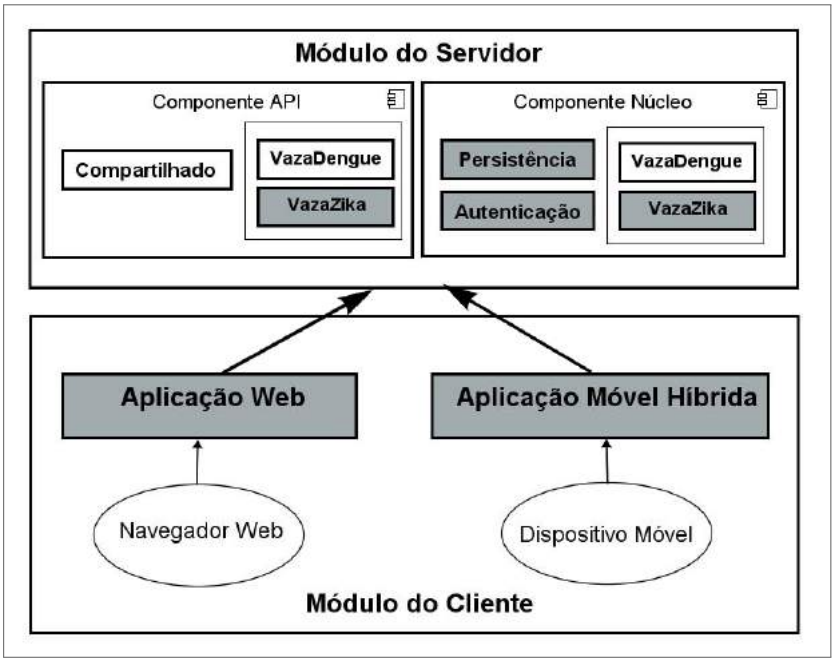

Figura 8 – Arquitetura do VazaZika

A Figura 8 mostra como ficou essa nova arquitetura. As partes destacadas em cinza denotam os componentes que sofreram alterações ou que foram introduzidos na nova arquitetura.

O novo fluxo de requisições, após a gamificação, segue o modelo descrito na Figura 9. Um jogador, ao desempenhar uma atividade na plataforma, faz uma requisição para a nova API VazaZika, a qual será interceptada pelo *Auth Interceptor*. Caso o jogador esteja autenticado na plataforma, após passar pelas operações do VazaDengue, sua requisição será interceptada pelo *Gamification Interceptor*, o qual é responsável por prover os serviços de gamificação, permitindo a contibilização de pontos, execução das regras e dar acesso aos demais elementos do jogo.

Fonte: Adaptado da apresentação do workshop do VazaZika

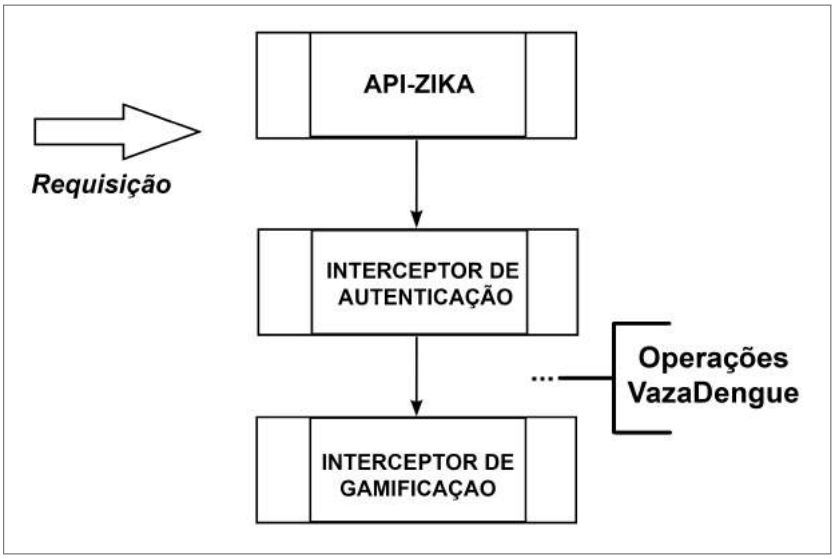

Figura 9 – Fluxo da Gamificação

3.2 NÚCLEO DA GAMIFICAÇÃO DO VAZAZIKA

Segundo foi apresentado na Figura 8, houve diversas alterações na arquitetura do então VazaDengue para atender aos requisitos de gamificação. Porém, este trabalho se concentrou em uma parte específica dessas alterações: o componente núcleo. Foi verificado que é neste componente que se encontram todas as regras de negócio e mecanismos relacionados à gamificação introduzida na plataforma. Por este motivo, a pesquisa tomou este ponto de partida para avaliar os impactos da gamificação na arquitetura existente.

Conforme a mesma Figura 8, ainda no módulo servidor, o componente API também sofreu alterações. Porém, ao ser investigado, foi possível perceber que as alterações não estavam relacionadas a mecanismos de gamificação em si, mas somente à criação de controladores que pudessem responder as requisições que partiam do módulo cliente. Dessa forma, estas requisições eram distribuídas para as áreas responsáveis correspondentes no componente núcleo e ali eram processadas as operações relacionadas ao jogo implementado na plataforma.

Tendo sido identificada a parte específica da arquitetura onde estavam implementados os elementos de gamificação, o próximo passo dado foi levantar questões de pesquisa que pudessem nortear o estudo dos impactos arquiteturais causados pela gamficação. Estas questões de pesquisa foram elaboradas a partir dos seguintes objetivos: descobrir se a arquitetura foi afetada negativamente durante o processo de gamificação da plataforma; caso tenha sido afetada negativamente, descobrir se os elementos de gamificação implementados estão diretamente realcionados a esses impactos.

A seguir as questões de pesquisa elaboradas são apresentadas, e mais adiante são detalhados os passos seguidos para responder a cada uma delas.

3.2.1 Questões de Pesquisa

Com o objetivo de descobrir se a arquitetura do então VazaDengue foi afetada negativamente pela introdução de gamificação na plataforma, as seguintes perguntas de pesquisa foram elaboradas:

- **QP1** - Qual era o estado da arquitetura da plataforma antes de ser gamificada e qual é o estado atual desta arquitetura?
- **QP2** - Quais métricas podem ser usadas para medir os impactos sofridos pela arquitetura existente, causados pela forma de gamificação escolhida?
- **QP3** - De acordo com as métricas escolhidas, a qualidade da arquitetura se manteve proporcional a evolução do sistema?
- **QP4** - Existe alguma relação entre os desafios percebidos pelos desenvolvedores e o impacto que o processo de gamificação causou à arquitetura?

Para responder a essas questões, foram seguidas algumas etapas de análise, as quais são descritas a seguir.

3.2.2 Impactos na Arquitetura

O primeiro passo para responder as questões de pesquisa foi dado através da investigação dos impactos na arquitetura do sistema ao longo do processo de gamificação. Algumas opções de ferramentas de análise estática arquitetural foram avaliadas em termos de métricas fornecidas, suporte de documentação oferecido e meios de extração dessas métricas, para obter as respostas dessas questões. Após a identificação das ferramentas necessárias para o estudo, foram aplicadas algumas das métricas mais utilizadas no meio acadêmico e na indústria para mensurar o tamanho do sistema. Essas métricas foram calculadas antes, durante e depois da gamificação. São elas: número de linhas de código e número de classes ou arquivos Java, uma vez que a implementação foi realizada utilizando esta linguagem.

Em seguida, foram identificadas as métricas relacionadas à qualidade da arquitetura. Segundo (LAKOS, 1996), um dos mais importantes aspectos de qualidade da arquitetura de um software está ligado aos níveis de acoplamento entre os componentes deste software. Sendo assim, (LAKOS, 1996) introduziu as métricas *DependsOn*, *Cumulative Component Dependency* (CCD) e *Average Component Dependency* (ACD) como meio de quantificar os níveis de acoplamento de um sistema.

A métrica *DependsOn* descrita por (LAKOS, 1996) se refere ao número de componentes dos quais o componente que está sendo avaliado depende. No nosso caso, um componente é entendido como um arquivo com extensão Java, por isso, a partir do próximo capítulo estes componentes serão chamados de classes. Se no arquivo Java sendo avaliado existir

algum tipo de referência a outro arquivo Java, então podemos dizer que o primeiro arquivo depende do segundo. E conforme foi explicado no Capítulo 2, o valor da métrica *DependsOn* é igual ao número de dependências que o componente avaliado possui mais um, pois ele também depende dele mesmo.

Nos casos das métricas *Cumulative Component Dependency* (CCD) e *Average Component Dependency* (ACD), estas são calculadas a partir da métrica *DependsOn*. O somatório de todas dependências presentes no sistema formam o número cumulativo de dependências, e a divisão desse valor pela quantidade de componentes presentes no sistema dão o resultado da média de dependências por componente do sistema como um todo.

A seguir descrevemos mais detalhadamente sobre o processo de análise de algumas ferramentas que poderiam fornecer os cálculos dessas métricas, explicando o porquê delas terem sido escolhidas para avaliar os impactos na arquitetura advindos do processo de gamficação, e sobre a descrição de outras métricas fornecidas por estas ferramentas.

3.2.2.1 Análise de ferramentas

Após algumas pesquisas, as ferramentas STAN (STAN, 2019) e SonarGraph Architect (SONARGRAPH, 2019) foram escolhidas como potenciais instrumentos que auxiliariam na investigação dos aspectos quantitativos e qualitativos da arquitetura antes, durante e após a gamificação. As duas ferramentas foram analisadas sob o ponto de vista de vantagens e desvantagens de cada uma e se ambas seriam necessárias, ou se apenas uma delas já seria suficiente para identificação dos impactos na arquitetura. A seguir está descrito o processo de avaliação dessas ferramentas.

1. STAN

   A ferramenta STAN foi a primeira avaliada neste estudo. Esta foi construída para análise de projetos de software especificamente escritos na linguagem Java. Ela possui um conjunto de métricas mostrado na Tabela 3, os quais avaliam aspectos de qualidade arquitetural, tais como acoplamento e coesão do sistema, ciclicidade e complexidade ciclomática.

   A ferramenta foi colocada em execução, a princípio, para avaliar apenas a arquitetura da plataforma antes e depois de ser gamificada. A intenção era de obter um direcionamento para o estudo, ao observar os valores obtidos em cada situação. Já era esperado que houvesse uma elevação considerável nas métricas relacionadas ao tamanho do projeto, como por exemplo *Number of Packages, Number of Units, Estimated Lines of Code* entre outras, já que houve implementação de código. Entretanto, se houvesse uma desproporcionalidade de crescimento em métricas relacionadas à qualidade, tais como *AVG Component Dependency between Packages, AVG Component Dependency between Units, Cyclomatic Complexity*, entre outras, seria

Tabela 3 – Métricas Ferramenta STAN

Number of Packages	Number of Libraries
Methods/Class	Number of Units
Units/Package	Fields/Class
Estimated Lines of Code	AVG Cyclomatic Complexity
Estimated Lines of Code per Top Level Class	Fat for Flat Package Dependencies
AVG Component Dependency between Packages	Fat for Top Level Class Dependencies
AVG Component Dependency between Units	AVG Distance
AVG Weighted Methods per Class	AVG Absolute Distance
AVG Depth of Inherintance Tree	AVG Number of Children
AVG Coupling between objects	AVG Response for a Class
AVG Lack of Cohesion in Methods	

Fonte: Próprio autor

por este indicativo que o estudo iria ser direcionado. O princípio de desproporcionalidade adotado diz respeito à comparação entre o aumento de tamanho do sistema e o aumento dos valores dessas métricas de qualidade.

Depois dessa análise, foi feita uma avaliação das vantagens e desvantagens da utilização da ferramenta STAN neste estudo, em termos de disponibilidade da ferramenta, meios de extração dos resultados das métricas calculadas e documentação disponível. Estes fatores iriam colaborar para o bom andamento do estudo, no sentido de fornecer, de maneira prática, meios de se chegar aos resultados pretendidos, a fim de se obter as respostas para as perguntas de pesquisa elaboradas.

Dentre as vantagens de utilizar a ferramenta STAN podemos citar o fato dela ser gratuita, desde que o projeto não ultrapasse o número de 500 classes. O fato do objeto de estudo ter sido escrito predominantemente na linguagem Java, também acabou por ser contado como um dos potenciais motivos de uso da ferramenta. Outra vantagem se deve a possibilidade de integração com a IDE Eclipse, o que poderia facilitar a análise do projeto, já que este foi construído nesta IDE. Por fim, a ferramenta está disponível como uma aplicação *standalone*, dando maior flexibilidade a projetos que não foram criados em uma IDE específica.

Por outro lado, a ferramenta STAN também apresentou algumas desvantagens, como por exemplo, ao ser comparada com a segunda ferramenta, SonarGraph, foi possível perceber que a documentação de apoio ao uso da ferramenta STAN não era tão completa quanto a da segunda. Isto poderia dificultar a utilização durante o processo de extração dos resultados.

Outro ponto observado como desvantagem, também ao ser comparada com a ferramenta SonarGraph, se deve ao fato do conjunto de métricas disponível em STAN

ser bem menor. Outra desvantagem foi notada, ao perceber que a ferramenta STAN disponibilizava unicamente a visualização das dependências dos componentes, enquanto que na outra ferramenta é possível criar regras de violação da arquitetura.

Por fim, a impossibilidade de extração das métricas de maneira a facilitar a análise dos valores por meio de uma planilha, por exemplo, dificultaria na agilidade da obtenção dos resultados.

2. SonarGraph

Em seguida, a ferramenta SonarGraph foi avaliada. Ela também possui um conjunto de métricas que avaliam os aspectos de qualidade arquitetural, tais como acoplamento, coesão e ciclicidade. A Tabela 4 apresenta estas métricas.

Tabela 4 – Métricas Ferramenta SonarGraph

AVG Component Dependency (ACD)	Biggest Component Cycle Group
Biggest Package Cycle Group	Byte Code Instructions
Cumulative Component Dependency (CCD)	Code Comment Lines
Comment Lines	Component Dependencies to Remove
Component Dependencies to Remove	Cyclicity
Highest ACD	Lines of Code
Maintainability Level	Normalized CCD
Number of Code Duplicates	Number of Component Cycle Groups
Number of Components	Number of Cyclic Components
Number of Cyclic Packages	Number of Duplicated Code Lines
Number of Modules	Number of Package Cycle Groups
Number of Packages	Number of Statements
Parser Dependencies to Remove	Propagation Cost
Relative ACD	Relative Cyclicity
Relative Cyclicity	Source Element Count
Structural Debt Index	Total Lines

Fonte: Próprio autor

Dentre as vantagens de utilizar a ferramenta SonarGraph podemos citar o fato dela possuir licença gratuita para propósitos acadêmicos, não se limitando ao número de classes do projeto. Possui também um conjunto de métricas maior que o da ferramenta STAN, englobando a maior parte das métricas desta, deixando de fornecer, apenas, algumas variações de métricas relacionadas ao tamanho do sistema e uma métrica relacionada a complexidade ciclomática, segundo foi apresentado na Tabela 3. Podemos citar ainda a disponibilidade da ferramenta como uma aplicação *standalone*, dando maior flexibilidade a projetos construídos em diferentes IDEs.

A ferramenta fornece ainda a possibilidade do gerenciamento de problemas arquiteturais do projeto, através de uma Linguagem Específica de Domínio (DSL). Possui scripts baseados em Groovy de métricas pré-calculadas, possibilitando a customização e criação de novas métricas.

A documentação disponível fornece um manual (SONARGRAPH, 2019) com diversos capítulos abordando detalhadamente cada funcionalidade da ferramenta, além de fornecer guias e exemplos para os que desejam manipulá-la. A ferramenta ainda dispõe de integração com outras ferramentas de qualidade de software, como por exemplo SonarQube (SONARSOURCE, 2013).

Por fim, SonarGraph dispõe da possibilidade de exportação dos valores das métricas calculadas para arquivos com extensão xls, podendo ser manipulados em planilhas Excel. A única desvantagem percebida na ferramenta SonarGraph está relacionada à curva de aprendizado. Por se tratar de uma ferramenta com muitas funcionalidades, e ainda dispor de uma linguagem de domínio específica (DSL), com possibilidades de criação de scripts Groovy para calcular novas métricas, o tempo necessário de estudo para dominar a ferramenta completamente é bem mais longo comparado ao da ferramenta anterior. No entanto, para o presente estudo, não foi necessário utilizar muitas funcionalidades da ferramenta, uma vez que o objetivo seria apenas entender a possível erosão estrutural (MARTIN, 2002) do sistema durante o processo de gamificação.

Após a análise das duas ferramentas, foi possível perceber que uma se apresenta com mais vantagens que a outra, neste caso a ferramenta SonarGraph. No entanto, também foi notado que esta ferramenta não dispõe da métrica *Cyclomatic Complexity* presente na ferramenta STAN, a qual está atrelada aos conceitos de qualidade de software, uma vez que esta métrica serve para mensurar a complexidade de um determinado módulo (uma classe, um método, uma função etc), a partir da contagem do número de caminhos independentes que ele pode executar até o seu fim. [1] Porém, ao ser calculada na ferramenta STAN, os valores desta métrica apresentados na condição do sistema não gamificado e gamificado quase não sofreram alteração. Isto indica que, apesar do software ter aumentado seu tamanho, o número de fluxos alternativos de execução no sistema não aumentaram significativamente. Mais detalhes deste relato são discutidos no Capítulo 4.

Após considerar todas as vantagens e desvantagens das ferramentas, além do atendimento às necessidades do presente estudo, a ferramenta escolhida para realizar a análise da qualidade da arquitetura do sistema foi a SonarGraph.

[1] https://www.treinaweb.com.br/blog/complexidade-ciclomatica-analise-estatica-e-refatoracao/

3.2.2.2 Métricas de Tamanho do Sistema

Após a escolha da ferramenta de apoio para analisar a arquitetura do sistema antes, durante e após o processo de gamificação, foram avaliadas algumas métricas que poderiam indicar o quanto o sistema evoluiu durante este processo e assim, futuramente, analisar as proporções de impacto que a arquitetura sofreu com todo este processo.

A ferramenta SonarGraph fornece uma série de métricas que calculam aspectos de tamanho do sistema, dentre elas podemos citar: *Number of Components*, *Number of Packages*, *Lines of Code*, *Number of Statements*, *Number of Modules* e *Total Lines*.

A métrica relativa ao número de linhas de código (LOC) foi uma das adotadas nesse estudo, por se tratar de uma métrica muito utilizada tanto na industria, quanto na academia, para medir o tamanho e custo de implementação de um software (NGUYEN et al., 2007). Já a métrica referente ao número de componentes também foi adotada, devido a natureza de análise qualitativa da arquitetura futuramente discutida nesse estudo. Um componente é a menor unidade de design físico de um sistema, segundo (LAKOS, 1996). No presente estudo, um componente corresponde a um arquivo com extensão ".java", ou seja, será tratado como uma classe Java.

A análise dos resultados dessas duas métricas se deu através da aplicação de seus respectivos cálculos na versão do sistema antes de ser gamificado e após ser gamificado. Esse procedimento forneceria os dados necessários para comparar a evolução do sistema em termos de tamanho e em termos de qualidade futuramente.

3.2.2.3 Métricas de Qualidade da Arquitetura

Depois da análise das dimensões do projeto VazaZika antes, durante e depois da gamificação, era necessário então analisá-lo sob o ponto de vista das métricas que avaliam os aspectos qualitativos da arquitetura.

(MARTIN, 2002) descreveu de maneira conceitual alguns sintomas conhecidos que podem ajudar a descobrir se um determinado software tem problemas arquiteturais, são eles:

- Rigidez: o sistema é difícil de mudar, porque qualquer mudança força a necessidade de uma série de outras mudanças.
- Fragilidade: mudanças fazem o sistema quebrar em partes conceitualmente não relacionadas.
- Imobilidade: é difícil separar o sistema de forma a fornecer componentes reusáveis.
- Viscosidade: fazer as coisas incorretamente é mais fácil que fazê-las corretamente.
- Opacidade: o código é difícil de ler e entender. Ele não consegue expressar bem a sua utilidade.

Sistemas com altos níveis de acoplamento, ou seja, quando o número de componentes que dependem de outros é alto em relação as proporções de tamanho do sistema, costumam apresentar esses sintomas com facilidade. Sendo assim, para avaliar a qualidade da arquitetura do VazaZika, foi preciso analisá-lo sob o ponto de vista das métricas que avaliam os níveis de acoplamento do sistema.

As métricas relativas a média de dependências por componente (ACD) e número cumulativo de dependências dos componentes (CCD) do sistema foram calculadas e avaliadas. Neste passo, o número de componentes envolvidos em ciclos também foi calculado e avaliado, pois tinha relação com os níveis de acoplamento do sistema, influenciando os resultados das duas outras métricas ora mencionadas.

3.2.3 Relação entre Desafios e Impactos

Para investigar as possíveis relações entre os impactos identificados na seção anterior e os resultados obtidos a partir do estudo dos desafios enfrentados pelos desenvolvedores durante a gamificação do sistema, os grupos de desafios da Tabela 1 foram revisitados.

Ao todo, 4 dos 8 grupos de desafios identificados foram incluídos no estudo, por se apresentarem como fortes candidatos a demonstrarem relação direta com os impactos na arquitetura do sistema VazaZika. A seguir serão mostrados, de forma mais detalhada, os motivos que fizeram certos grupos de desafios serem incluídos no presente estudo e os motivos que excluíram os outros.

3.2.3.1 Grupos de Desafios Incluídos

Dentre os grupos de desafios incluídos temos os seguintes: [GRP01], [GRP04], [GRP05] e [GRP08]. A seguir, o detalhamento de cada um deles.

- **[GRP01] Definir o sistema gamificado.** Este grupo foi incluído, pois traz as atividades desafiadoras identificadas pelos desenvolvedores no momento de decidir a organização do código fonte, decomposição dos serviços e definição da camanda de persistência dos dados. Estes fatores estão fortemente ligados à arquitetura do sistema.

- **[GRP04] Implementar o sistema gamificado.** Este grupo foi selecionado, por conta das atividades relacionadas a implementação do código do sistema, o que influencia de maneira direta no resultado da arquitetura final.

- **[GRP05] Entender o sistema legado.** Este grupo, por sua vez, foi escolhido devido às atividades reconhecidas como desafiadoras em relação ao entendimento da arquitetura preexistente.

- **[GRP08] Testar o sistema gamificado.** Por fim, este grupo foi selecionado, por causa das atividades de testes, as quais são implementadas no sistema e sofrem influência da modelagem arquitetural deste.

3.2.3.2 Grupos de Desafios Excluídos

Dentre os grupos de desafios excluídos temos os seguintes: [GRP02], [GRP03], [GRP06] e [GRP07]. A seguir, estes também serão detalhados.

- **[GRP02] Conceber a arte e o design do sistema.** Este grupo foi excluído, pois as atividades desafiadoras identificadas pelos desenvolvedores estão relacionadas aos aspectos visuais do sistema, tais como cores e layout, não estando ligadas diretamente aos aspectos arquiteturais do sistema.
- **[GRP03] Especificar o sistema gamificado.** Este grupo foi excluído, pois as atividades desafiadoras têm relação com a escolha dos elementos de jogo e das regras, ou seja, de aspectos do estudo da gamificação em si. Os desafios descritos aqui dizem respeito as dificuldades enfrentadas ao decidirem aderir ou descartar determinado elemento de jogo. Portanto, não tem relação direta com a arquitetura do sistema.
- **[GRP06] Gerenciar os times de desenvolvimento.** Este grupo, por sua vez, não foi selecionado devido às atividades reconhecidas como desafiadoras estarem relacionadas apenas ao gerenciamento do projeto.
- **[GRP07] Comunicação dos times de desenvolvimento.** Por fim, como a própria descrição aponta, este grupo reune as atividades desafiadoras ligadas aos aspectos de comunicação entre os membros das equipes. Portanto, não tem ligação com a arquitetura do sistema e por isso foi excluído.

3.2.4 Análise do Código Fonte

Após todo o processo de análise da arquitetura do sistema através das ferramentas e identificação dos desafios que apresentavam possíveis relações com os impactos percebidos na arquitetura, foi realizada uma análise mais aprofundada nesta. Dessa vez, observando o código fonte do sistema, para investigar na prática como funcionavam os relacionamentos descobertos através da análise visual, com a ajuda da ferramenta SonarGraph.

Essa análise do código do sistema tinha por objetivo identificar oportunidades de mudanças, sem causar prejuízo às regras e lógicas implementadas, a fim de se obter um estado de qualidade melhor da arquitetura, no sentido de diminuir os níveis de acoplamento e ciclicidade, caso fosse possível.

Como a investigação resultou em oportunidades de mudança no código, aparentemente sem infringir as regras e lógicas implementadas, as mudanças foram então efetuadas. Esta

ação tinha por objetivo melhorar os níveis de acoplamento do sistema. Logo em seguida as métricas foram recalculadas com o propósito de comparar o estado final da gamificação com o estado após as mudanças efetuadas.

3.3 RESUMO DO CAPÍTULO

Para descobrir os métodos adequados de investigar os impactos causados na arquitetura da plataforma ao ser gamificada, foi iniciada a análise de sua arquitetura antes e depois do processo de gamificação. Esta análise permitiu a descoberta de como a adaptação da arquitetura do VazaDengue foi realizada, para receber as funcionalidades de gamificação. Um componente núcleo ficou responsável por conter as regras de negócio da gamificação implementada, com isso a análise foi focada neste componente.

Em seguida, foram elaboradas questões de pesquisa para nortear o estudo dos impactos da gamificação na plataforma de software. Para responder a essas questões, duas ferramentas foram analisadas e executadas, a fim de obter um comparativo entre elas e descobrir qual delas melhor se enquadrava nos objetivos do presente estudo. A ferramenta SonarGraph se mostrou melhor adequada, devido suas vantagens em relação à ferramenta STAN.

Para avaliar os aspectos qualitativos da arquitetura antes e depois de ser gamificada, primeiro, métricas foram escolhidas, do ponto de vista de evolução do tamanho do sistema, com o objetivo de obter uma proporção de crescimento, a qual pudesse ser contrastada com a proporção de aumento ou diminuição da qualidade arqitetural. Em seguida, as métricas de qualidade foram escolhidas com base nos sintomas descritos por (MARTIN, 2002), os quais dão indícios de que um software tem problemas arquiteturais. Segundo ele, os níveis de acoplamento de um sistema estão fortemente relacionados à qualidade dele, sendo assim, métricas relativas a acoplamento e coesão foram escolhidas nesta fase do estudo.

Logo após, os grupos de desafios, identificados num estudo anterior, foram analisados com a finalidade de descobrir possíveis relações entre as declarações dadas pelos desenvolvedores durante o processo de gamificação e as descobertas feitas através da ferramenta de análise arquitetural. Dos oito grupos identificados, quatro deles foram cotados, porque mostraram alguma relação com os resultados obtidos na análise arquitetural e quatro foram excluídos, por não estarem diretamente ligados aos aspectos arquiteturais.

Por fim, foi feita uma análise diretamente no código fonte do sistema, a partir dos indicativos obtidos durante a análise visual da arquitetura da plataforma, através da ferramenta. Esta análise teve por objetivo identificar possíveis oportunidades de *refactoring*. Como a possibilidade de *refactoring* se mostrou factível, mudanças no código foram realizadas e as métricas foram recalculadas, produzindo assim novos índices para a qualidade da arquitetura.

No próximo capítulo serão abordados os resultados e discussões do processo apresentado neste capítulo.

4 RESULTADOS E DISCUSSÕES

Neste capítulo são descritos os resultados e discussões a respeito do estudo do VazaZika, uma plataforma de software gamificada, respondendo as questões de pesquisa mencionadas no capítulo anterior. O capítulo está estruturado da seguinte forma: a Seção 4.1 apresenta os resultados das métricas calculadas relativas ao tamanho e qualidade da arquitetura antes, durante e após o processo de gamificação; a Seção 4.2 apresenta os fatores que influenciaram os valores finais das métricas calculadas durante o processo de gamificação; a Seção 4.3 mostra os resultados das relações dos desafios escolhidos, descritos no capítulo anterior, com os impactos na arquitetura; por fim, a Seção 4.4 descreve os resultados da implementação de alguns refactorings no código do sistema, para fins de demonstração das conclusões obtidas a partir do estudo.

4.1 ESTADO DA ARQUITETURA ANTES E DEPOIS DA GAMIFICAÇÃO

4.1.1 Métricas de Tamanho

A Figura 10 mostra o total de linhas de código do sistema antes de ser gamificado no lado esquerdo e ao lado direito podemos observar o número de linhas de código do sistema após a gamificação.

Antes de ser gamificado, o sistema possuia um total de 6917 linhas de código e passou a ter 10087 linhas de código após esse processo. A maior parte dessas linhas foram

Fonte: Próprio autor

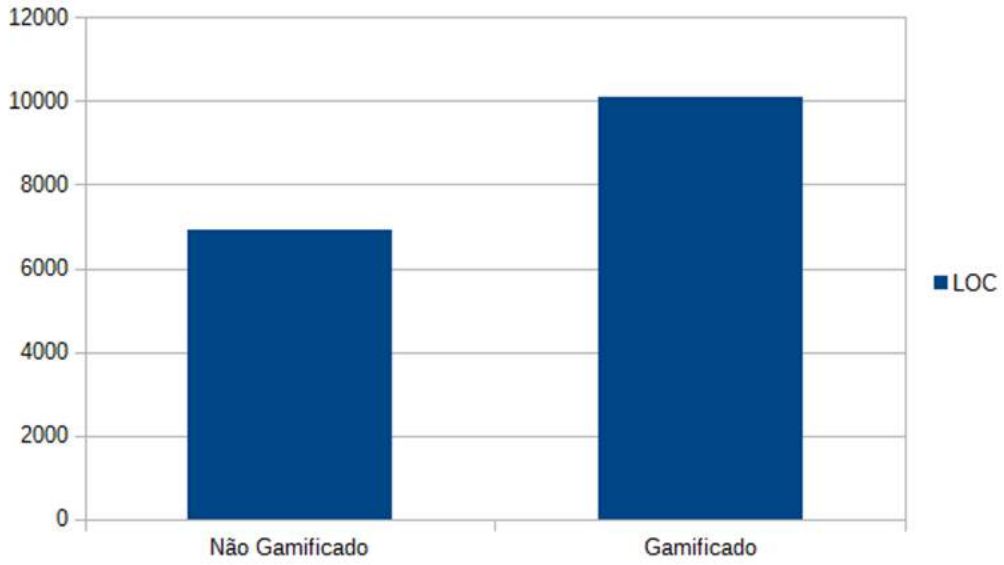

Figura 10 – Número de Linhas de Código Antes e Depois da Gamificação

Fonte: Próprio autor

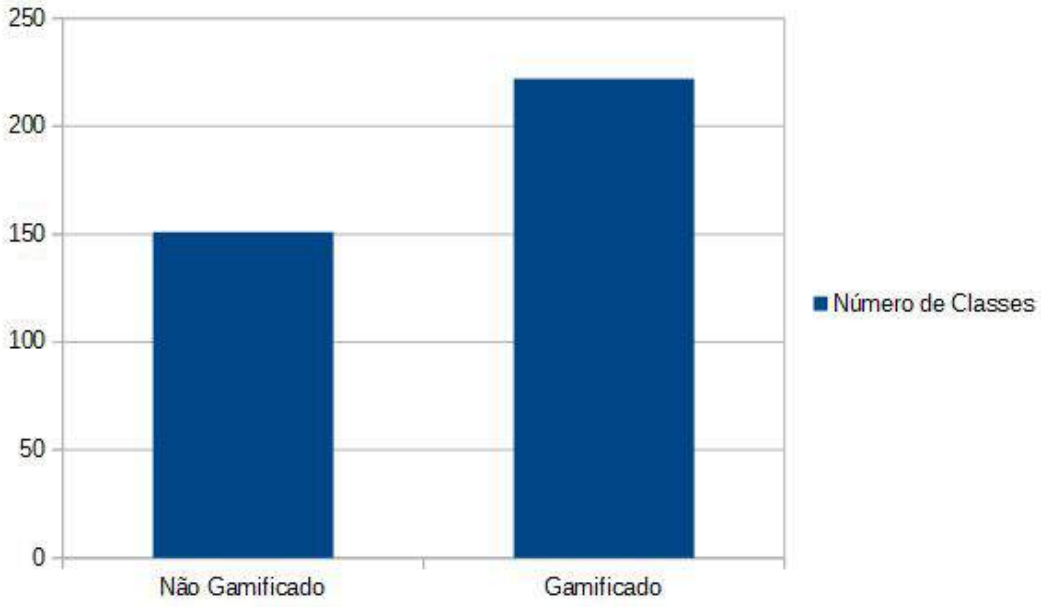

Figura 11 – Número de Classes Antes e Depois da Gamificação

introduzidas como consequência do processo de gamificação. Esta conclusão foi tirada a partir da percepção da adição de 16 novas entidades, especificamente relacionadas ao jogo implementado na plataforma, e da introdução de 5 novos serviços, também relacionados a gamificação. Além desses, foram introduzidas classes para gerenciar repositórios (14), classes de tratamento de exceção e validadores (3) e ainda outras classes para gerenciar restrições, as quais não foram investigadas em detalhes, por não se tratar do foco da pesquisa.

A Figura 11 demonstra a evolução do sistema em termos de número de classes, ou seja, quantidade de arquivos com extensão ".java". Mais uma vez o gráfico mostra do lado esquerdo o número de classes do sistema antes de ser gamificado e ao lado direito o número após a gamificação. O sistema possuia o total de 151 classes antes da gamificação e passou a ter 222 classes após este processo. Ao verificarmos o local no qual estas classes adicionais estavam inseridas no projeto, foi constatado que 62 delas pertenciam a um pacote Java chamado *gamification*. As 9 classes que faltavam para completar o total de 71 classes adicionadas ao projeto estavam divididas entre repositórios (3), utilitários (3) e novas entidades (3).

Apesar dessas classes não estarem no pacote *gamification* mencionado acima, isso não quer dizer que essas classes não tenham relação com o processo de gamificação. Na verdade, seria mais coerente pensar que as classes foram criadas devido às necessidades advindas do processo de gamificação do que por qualquer outro motivo. Somente de posse do código dessas classes não há como afirmar com toda certeza, mas ao que tudo indica, para os desenvolvedores fazia mais sentido colocar essas classes em outros pacotes Java já

Fonte: Próprio autor

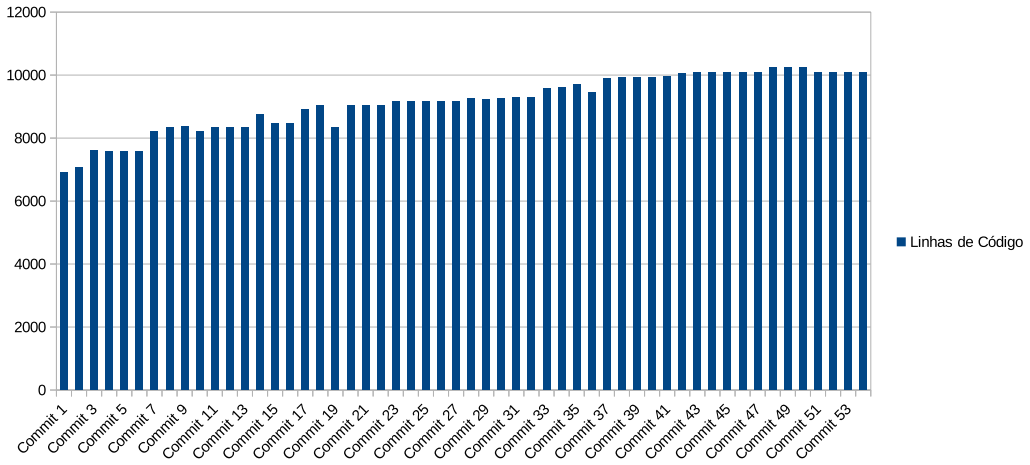

Figura 12 – Número de Linhas de Código ao Longo do Processo de Gamificação

Fonte: Próprio autor

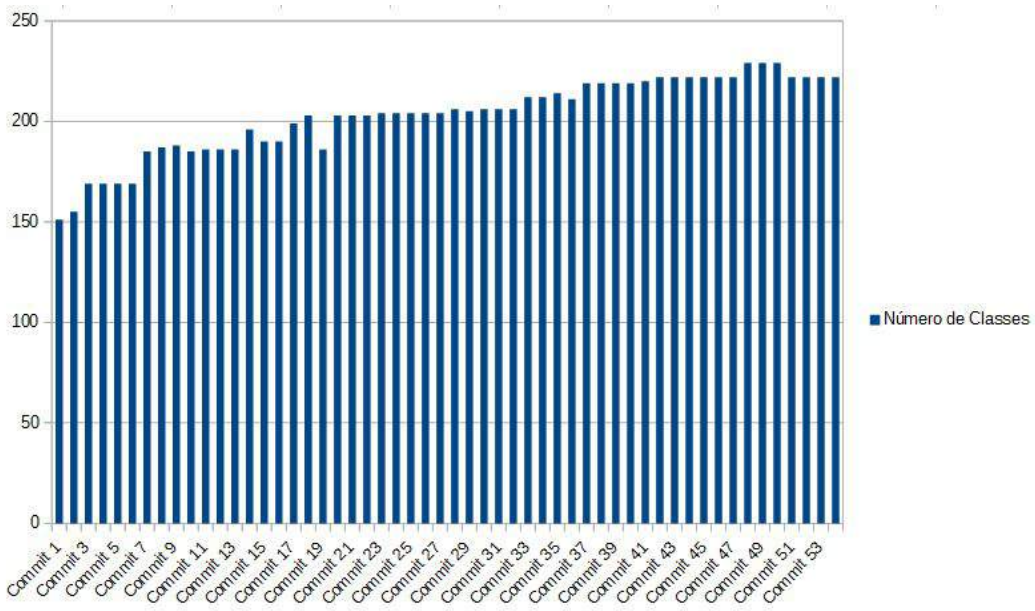

Figura 13 – Número de Classes ao Longo do Processo de Gamificação

existentes antes do processo de gamificação do que juntá-las às demais classes no pacote *gamification*.

Uma análise do processo de gamificação feita commit a commit mostra que as métricas de número de linhas de código e número de classes se comportaram de maneira muito semelhante durante toda a gamificação do sistema, o que já era esperado devido a natureza delas. Nas Figuras 12 e 13 é possível perceber mais claramente esta semelhança, ao serem colocados os dois gráficos próximos um do outro.

Fonte: Próprio autor

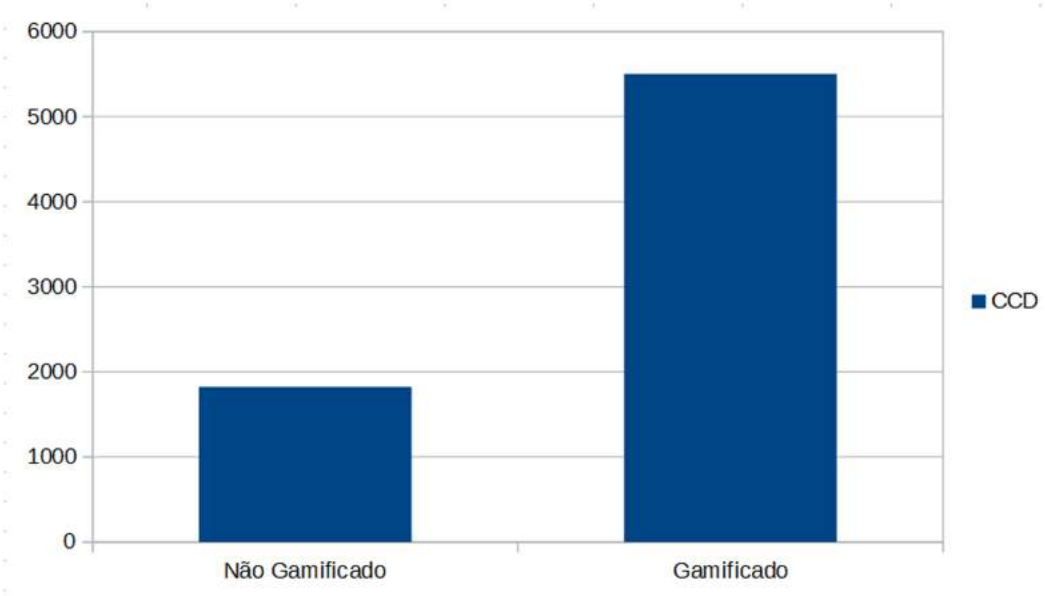

Figura 14 – Número Cumulativo de Dependências das Classes

4.1.2 Métricas de Qualidade

Semelhantemente ao processo realizado com as métricas de dimensionamento do sistema VazaZika, foram feitas análises em relação às métricas de níveis de acoplamento e ciclicidade do sistema antes, durante e depois do processo de gamificação.

A Figura 14 apresenta o número cumulativo de dependências das classes do sistema antes e depois da gamificação, à esquerda e à direita, respectivamente. No primeiro momento, o sistema apresentava o número de 1824 dependências cumulativas e passou a ter 5505 dependências cumulativas após o processo de gamificação.

A média de dependências das classes variou de 12.08 a 24.8 depois do processo de gamificação do sistema, conforme resultado mostrado na Figura 15. Já em relação ao número de classes envolvidas em ciclos, antes da gamificação, o sistema apresentava 24 classes nesse estado, quando este possuia um total de 151 classes no início da implementação da gamificação, e passou para o número de 47 clsses envolvidos em ciclos no final, de um total de 222 classes, conforme mostrado na Figura 16.

Uma análise dessas métricas feita a cada commit revela o comportamento delas ao longo do processo de gamificação. Essa abordagem abriu possibilidades de investigar, de maneira mais específica, quais mudanças afetaram o aumento ou diminuição dos níveis de acoplamento e ciclicidade durante o processo, dando um direcionamento mais eficaz no sentido de identificar os fatores que influenciaram os números apresentados.

A Figura 17 mostra o número cumulativo de dependências das classes por commit. É possível perceber que já na reta final do processo de gamificação, houve momentos em que esse número ultrapassou o valor de 5505, o qual foi idenficado ao calcular a métrica

Fonte: Próprio autor

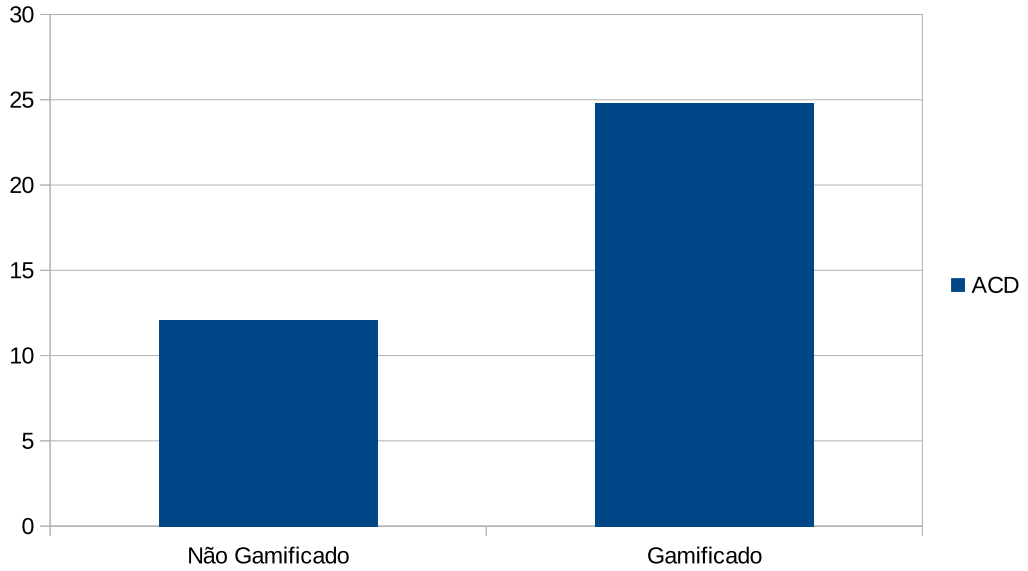

Figura 15 – Média de Dependências por Classe

Fonte: Próprio autor

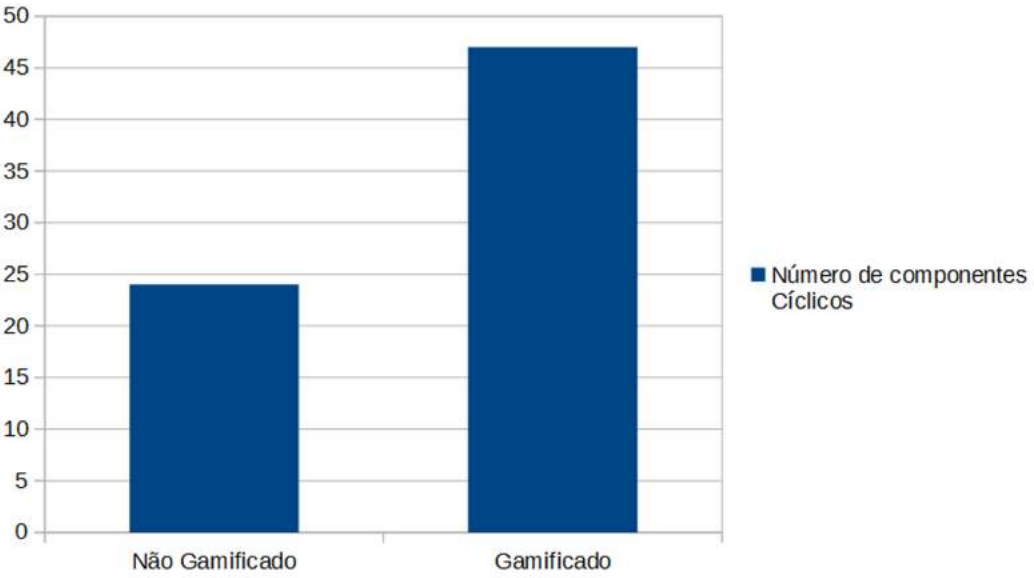

Figura 16 – Número de Classes Envolvidas em Ciclos

somente no estado final da implementação.

Ao calcular a média de dependências por classe commit a commit, o comportamento do gráfico gerado seguiu o mesmo padrão da métrica anterior. A Figura 18 mostra alguns commits específicos causando alguns picos, elevando a média de dependências das classes. Esses indicativos foram importantes para identificar as mudanças que causaram maior impacto na arquitetura do sistema durante o processo de gamificação.

No caso da métrica que calcula o número de classes envolvidas em ciclos, os valores

Fonte: Próprio autor

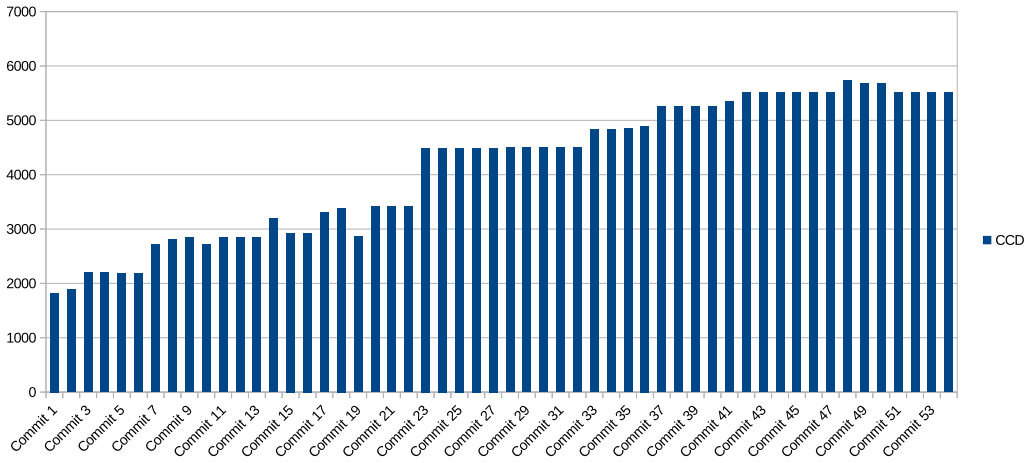

Figura 17 – Dependências Cumulativas Durante a Gamificação

Fonte: Próprio autor

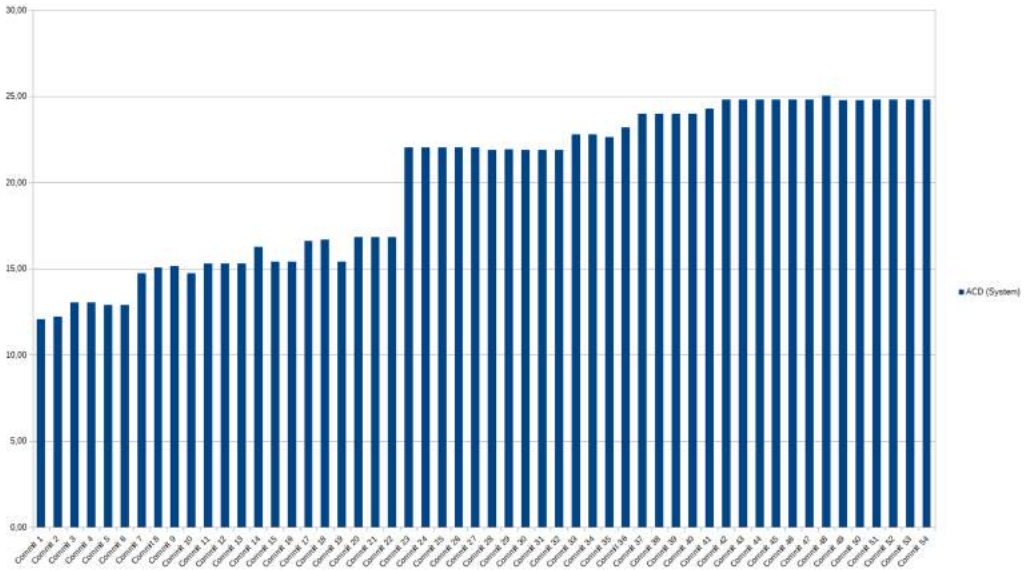

Figura 18 – Média de Dependências Durante a Gamificação

Fonte: Próprio autor

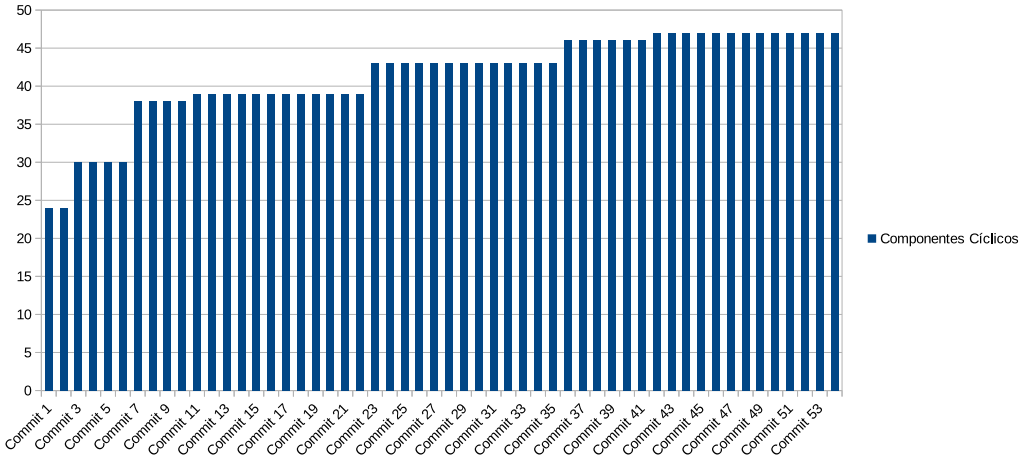

Figura 19 – Classes Envolvidas em Ciclos Durante a Gamificação

extraídos a partir de cada commit revelam que, ao contrário das métricas concernetes aos níveis de acoplamento, estes só aumentaram do início ao fim do processo de gamificação, conforme mostrado na figure 19. Estes foram, portanto, os impactos causados na arquitetura do sistema durante o processo de gamificação, de acordo com as métricas de qualidade calculadas.

Os valores obtidos a partir do cálculo das métricas de qualidade são, no mínimo, intrigantes, uma vez que em termos de tamanho, o sistema cresceu menos de 50%, enquanto que em termos de número de dependências cumulativas cresceu mais de 300%. Ao observar o número médio de dependências por classe, o aumento também apresenta uma discrepância considerável, já que o aumento foi de mais de 100%. Na seção a seguir serão apresentados alguns fatores que podem ter causado esses resultados.

4.2 FATORES DE INFLUÊNCIA

O modo como a gamificação foi introduzida na arquitetura do então VazaDengue influenciou bastante nos números obtidos através das métricas de qualidade calculadas. O fato de todas as requisições do jogador passarem por um interceptador fez com que este elevasse o número de dependências geral do sistema, uma vez que ele é o responsável por distribuir todo o fluxo de gamificação, acumulando em si as dependências de todos as outras classes relacionadas ao jogo.

4.2.1 Níveis de Acoplamento

A Figura 20 mostra um ranking das classes com mais dependências no sistema. Podemos então observar que a classe *GamificationInterceptor.java* ocupa o primeiro lugar do

ranking com o total de 71 dependências, seguido da classe *GamificationService.java* com 70 dependências.

Fonte: Próprio autor

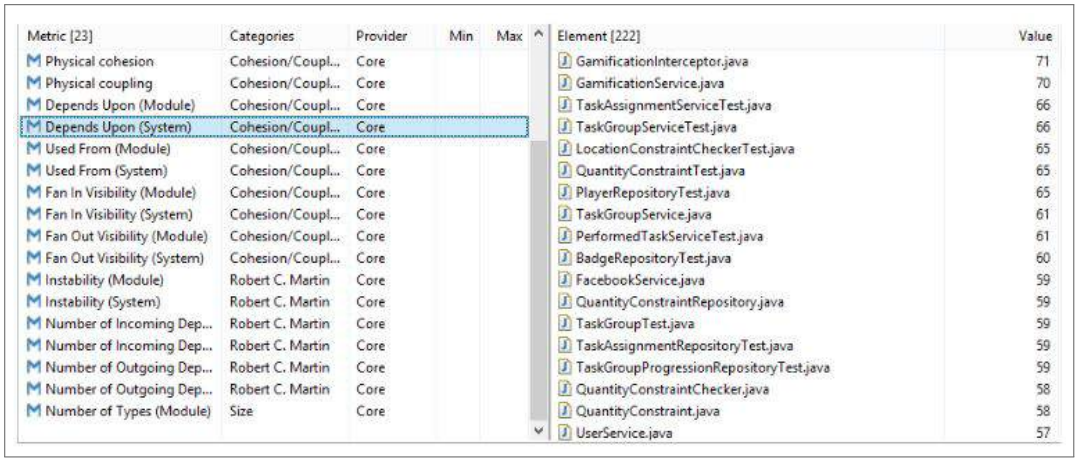

Figura 20 – Ranking de Classes com Mais Dependências

Analisando o grafo das dependências da classe *GamificationInterceptor.java*, foi possível identificar que apesar deste ocupar o topo do ranking, a única razão para tal resultado se dá pela ligação deste com a classe que ocupa a segunda posição do ranking. A classe *GamificationService.java* é a verdadeiro responsável por acumular em si as dependências das outras classes relacionados ao jogo. A Figura 21 mostra o grafo desta última classe com suas dependências diretas, apenas.

Fonte: Próprio autor

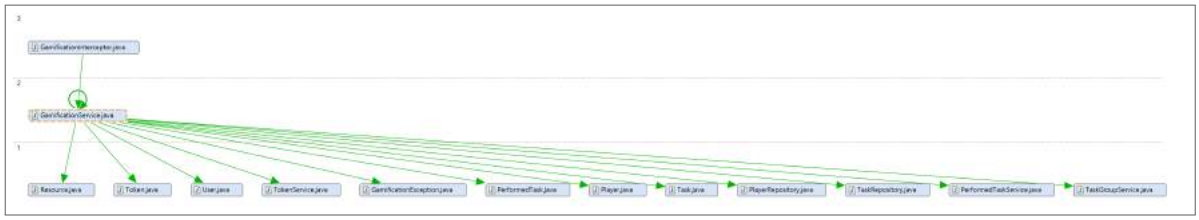

Figura 21 – Grafo *GamificationService*

Para identifcar com mais precisão os fatores que levaram a classe acima mencionada a elevar o número de dependências geral do sistema, foi necessário visualizar todo o cenário de dependências desta classe. A Figura 22 mostra o grafo de todas as dependências diretas e indiretas a partir da classe *GamificationInterceptor*.

Quando uma classe acumula em si tantas dependências de uma só vez, a tendência é aumentar os níveis gerais de acoplamento do sistema. Isso acontece no VazaZika, conforme foi constatado ao observar as dependências da classe *GamificationService*. Esta forma de implementar gamificação, na qual as requisições são processadas por um ponto centralizado na arquitetura, causa erosão arquitetural. O sistema apresentaria um melhor nível

de qualidade arquitetural se não houvesse tantas dependências acumuladas em uma única classe.

Fonte: Próprio autor

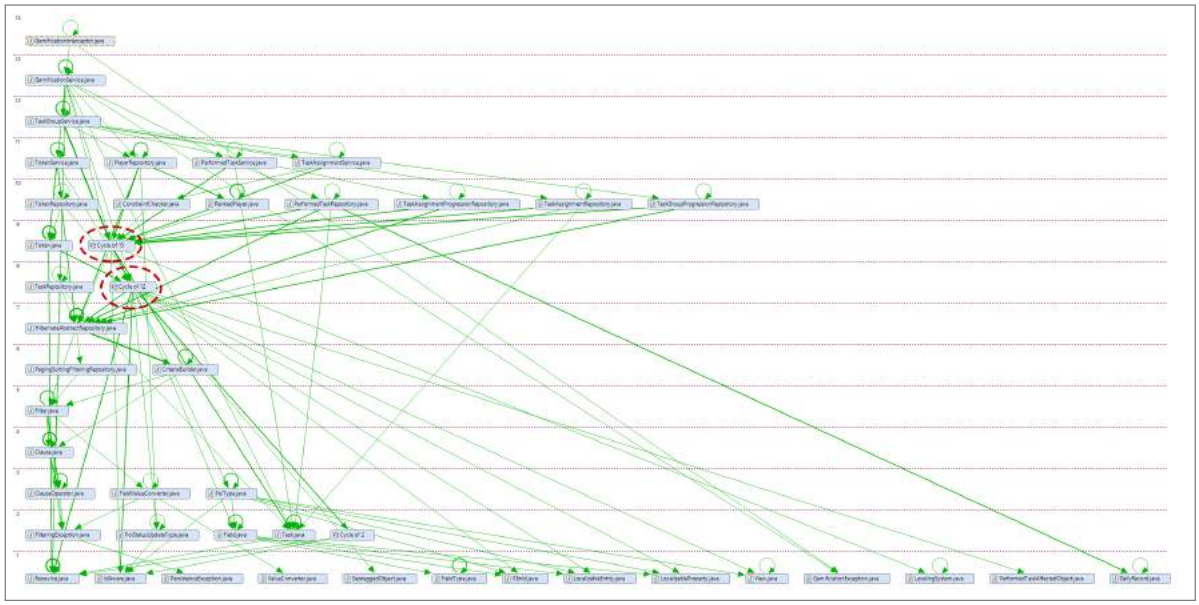

Figura 22 – Grafo Dependências Diretas e Indiretas do *GamificationInterceptor*

Com este cenário disponível, foi possível então perceber que havia dois notáveis grupos de classes envolvidas em ciclos. O primeiro comportando o total de 15 classes e um segundo contando com 12 classes em seu grupo, conforme destacado na figura.

4.2.2 Níveis de Ciclicidade

A Figura 22 mostra os dois grupos cíclicos destacados identificados no grafo. Foi necessário investigar quais elementos faziam parte de cada um desses grupos, para tentar descobrir os motivos que os levaram a fazer parte de estruturas cíclicas com estas dimensões. Esses dois grupos são detalhados a seguir.

A Figura 23 mostra todos as classes envolvidos no ciclo do primeiro grupo. Olhando de maneira superficial, é possível perceber que a maioria, senão todas as classes presentes neste grupo estão aparentemente relacionados ao jogo em si. Podemos observar no grupo cíclico mencionado a presença de classes nitidamente relacionadas ao jogo implementado no sistema, tais como *Player*, *Badge*, *Team*, *Progression*, *TaskGroup*, entre outras.

A Figura 24 mostra todas as classes envolvidas no ciclo do segundo grupo. Este, por sua vez, já não apresenta tantos indícios de que suas classes estão diretamente relacionados ao conteúdo do jogo introduzido à plataforma.

Entretanto, tendo relação ou não com as regras e elementos do jogo introduzidos no sistema, seria pelo menos negligência não avaliar se já havia este tipo de comportamento na arquitetura, mesmo antes da gamificação, ou se isso aconteceu somente após o início do

Fonte: Próprio autor

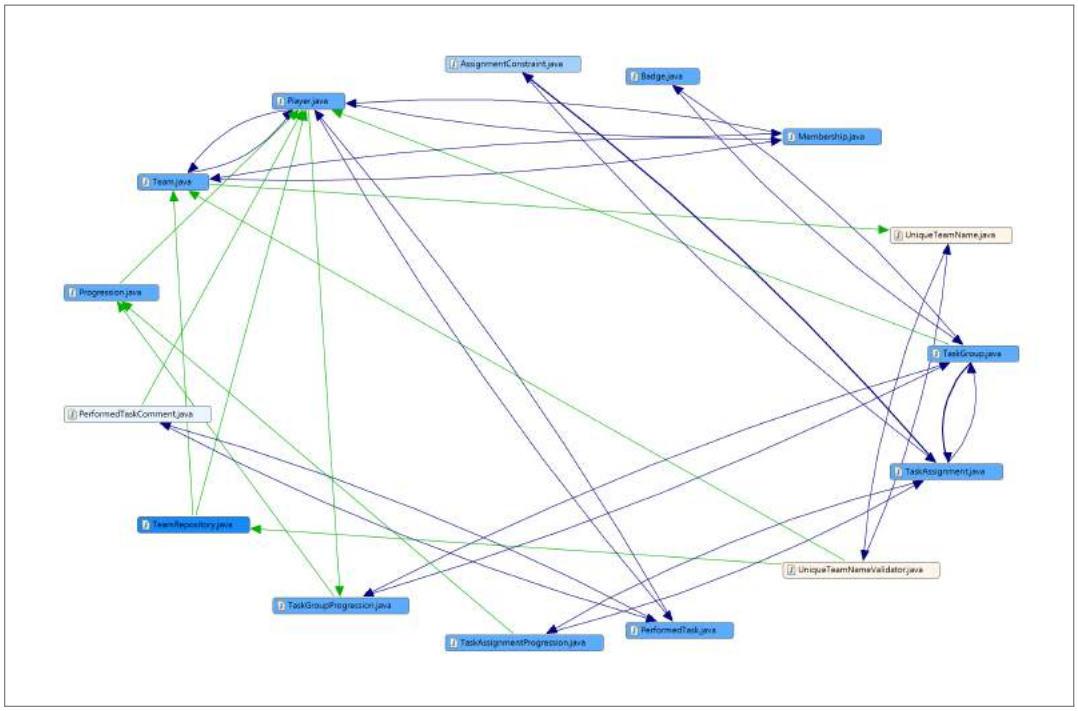

Figura 23 – Classes do Primeiro Grupo Cíclico

Fonte: Próprio autor

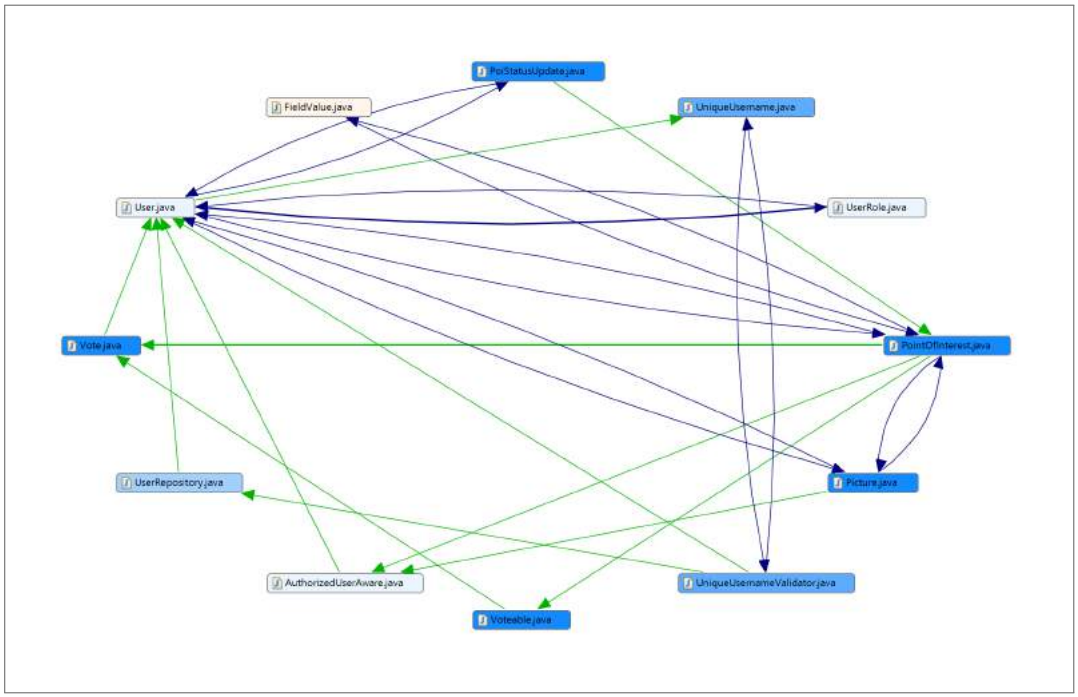

Figura 24 – Classes do Segundo Grupo Cíclico

processo. Por isso, todo o procedimento descrito acima foi repetido na versão do sistema antes de ser gamificado.

A Figura 25 mostra o ranking de classes com mais dependências, antes do sistema ser

Fonte: Próprio autor

Metric [23]	Categories	Provider	Min	Max	Element [151]	Value
Number of Violations (Co...	Architecture	Core			VoteService.java	41
Number of Violations (Par...	Architecture	Core			VoteRepositoryFactory.java	40
Component Rank (Module)	Code Analysis	Core			FieldValueMetaFieldsProvider.java	40
Component Rank (System)	Code Analysis	Core			PoiCommentVoteRepository.java	37
Fan In Maintainability Leve...	Cohesion/Coupl...	Core			FieldValueRepository.java	36
Fan In Maintainability Leve...	Cohesion/Coupl...	Core			PoiCommentRepository.java	36
Physical cohesion	Cohesion/Coupl...	Core			PointOfInterestRepository.java	36
Physical coupling	Cohesion/Coupl...	Core			PoiVoteRepository.java	36
Depends Upon (Module)	Cohesion/Coupl...	Core			PoiComment.java	35
Depends Upon (System)	Cohesion/Coupl...	Core			PoiCommentVote.java	35
Used From (Module)	Cohesion/Coupl...	Core			DenuntiationRepository.java	35
Used From (System)	Cohesion/Coupl...	Core			Denunciation.java	34
Fan In Visibility (Module)	Cohesion/Coupl...	Core			PoiVote.java	34
Fan In Visibility (System)	Cohesion/Coupl...	Core			PictureRepository.java	34
Fan Out Visibility (Module)	Cohesion/Coupl...	Core			Picture.java	33
Fan Out Visibility (System)	Cohesion/Coupl...	Core			PointOfInterest.java	33
Instability (Module)	Robert C. Martin	Core			FieldValue.java	33
Instability (System)	Robert C. Martin	Core			InstagramClientBuilder.java	27

Figura 25 – Classes do Segundo Grupo Cíclico Antes do Sistema Ser Gamificado

gamificado. Ao analisar o grafo com dependências diretas da classe que ocupa a primeira posição do ranking desta vez, obtemos o resultado expresso na Figura 26.

Fonte: Próprio autor

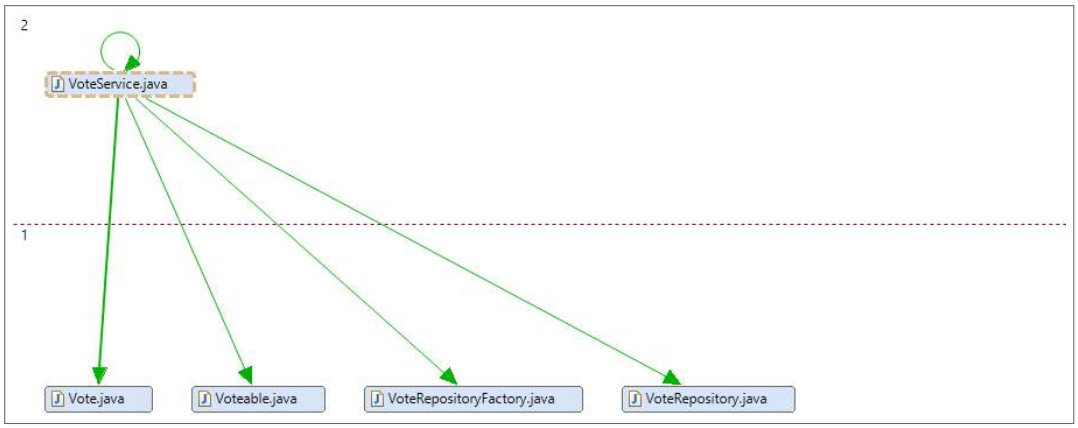

Figura 26 – Grafo de Dependências Diretas da Classe *VoteService*

Já no grafo contendo as dependências diretas e indiretas dessa mesma classe é possível identificar também um grupo de classes envolvidas em ciclo. Porém, o número de classes é bem menor, se comparado ao resultado anterior, apenas 5 classes. A Figura 27 mostra o grafo de dependências diretas e indiretas da classe e a Figura 28 mostra as classes envolvidos no ciclo identificado no grafo.

Dadas estas circunstâncias, é possível concluir que os índices de ciclicidade entre as classes do sistema antes da gamificação eram bem mais baixos que os dos posteriores ao processo. Portanto, essas classes envolvidas em ciclos foram importantes fatores de influência na diminuição da qualidade da arquitetura do sistema. A disposição das classes desta maneira na arquitetura pode ter sido influenciada pela tomada de decisão dos projetistas ao definirem o design do sistema.

Fonte: Próprio autor

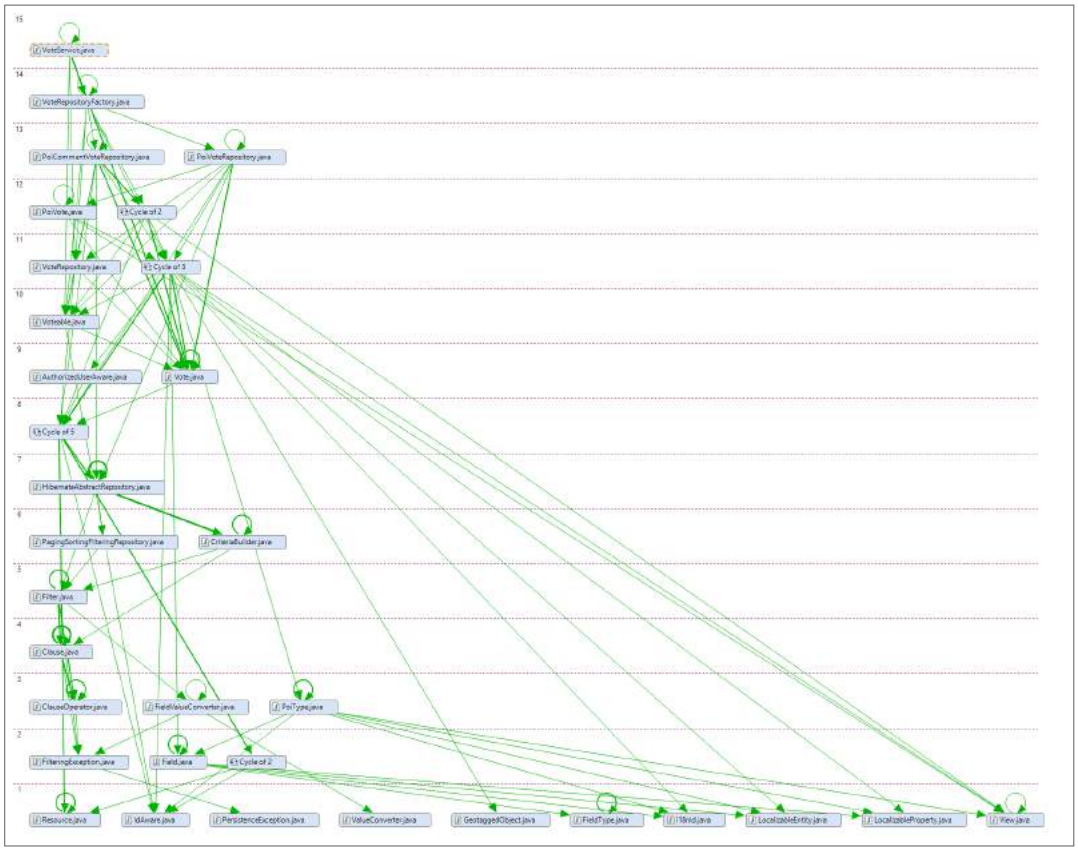

Figura 27 – Grafo de Dependências Diretas e Indiretas da Classe *VoteService*

Fonte: Próprio autor

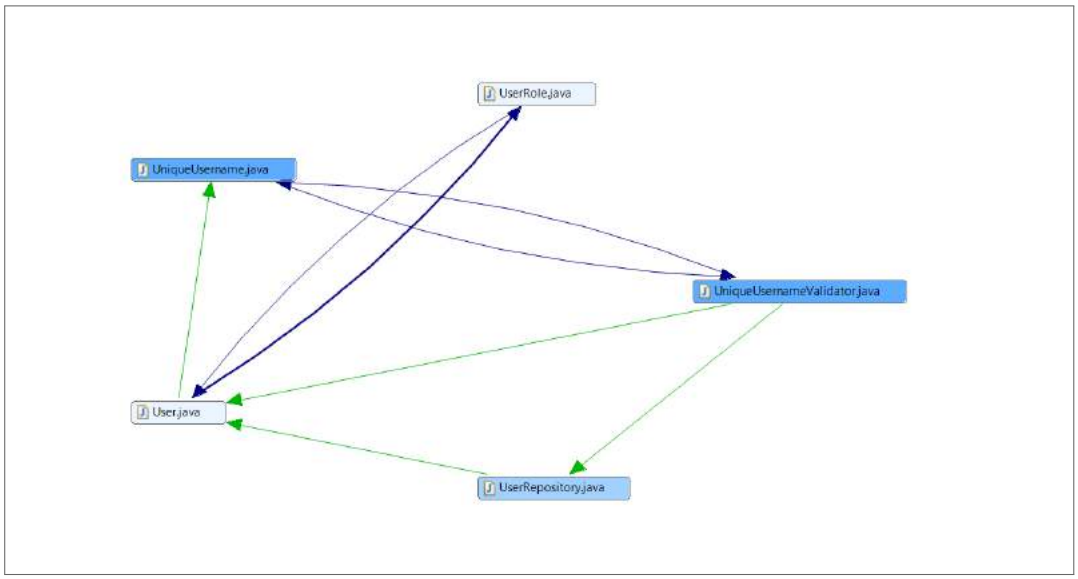

Figura 28 – Classes Envolvidas no Ciclo

4.3 DESAFIOS E ARQUITETURA

Como já mencionado no capítulo anterior, foram selecionados quatro dos oito grupos de desafios percebidos pelos desenvolvedores para identificação das possíveis relações com os impactos percebidos na arquitetura durante a gamificação do sistema.

Nesta seção são descritos os resultados das relações de cada um desses grupos e como essas relações foram identificadas.

4.3.1 Grupo 01 - Definir o Sistema Gamificado

O grupo 1 reune as atividades percebidas como desafiadoras em relação a definição do sistema gamificado. Este grupo foi caracterizado desafiador, na concepção dos desenvolvedores, devido às muitas limitações existentes na então plataforma VazaDengue. De fato, os próprios projetistas da antiga plataforma reconheceram que ela não foi designada para acomodar os serviços de gamificação.

Isso se tornou ainda mais evidente quando os elementos e regras do jogo definidos pelo time de gamificação não puderam ser totalmente endereçados ao time de desenvolvimento, devido a grande quantidade de mudanças que seriam necessárias. Inclusive, numa das entrevistas, um dos desenvolvedores mencionou que “o maior desafio foi reestruturar a infraestrutura da arquitetura para dar suporte a gamificação”.

Ainda no estudo dos desafios deste grupo, um dos fatores técnicos que levou os desenvolvedores a terem esse tipo de percepção está ligado à recorrente necessidade de mudanças na modelagem do banco de dados durante o processo de gamificação, para otimizar o gerenciamento das regras e seus respectivos elementos do jogo.

4.3.1.1 Uma Análise da Estrutura

Apesar dos esforços relatados sobre a reestruturação da arquitetura para acomodar a gamificação, principalmente em relação a modelagem das entidades do banco de dados, a análise feita sobre a arquitetura resultante revela que tais esforços não levaram em consideração a qualidade da arquitetura nos aspectos de acoplamento e ciclicidade. Ao observarmos as entidades do banco, principalmente aquelas relacionadas ao conteúdo do jogo introduzido à plataforma, podemos perceber que muitas delas estão não somente envolvidas em ciclos, como também apresentam ciclos diretos com outras entidades.

Como já visto na Figura 23, 12 das 15 classes presentes ali apresentam uma estrutura onde há um relacionamento de ciclidade direta, ou seja, quando uma classe tem uma flecha indo em direção a outra classe e outra flecha voltando desta mesma classe para a primeira.

Este caso também se repete no outro grupo cíclico mostrado na Figura 24. Ali, 8 das 12 classes apresentam este comportamento. E uma investigação mais detalhada revelou que todos eles, tanto os do primeiro grupo, quanto os do segundo, são entidades

do banco de dados, exceto quatro classes: *UniqueUsername*, *UniqueUsernameValidator*, *UniqueTeamName*, *UniqueTeamNameValidator*.

4.3.1.2 Uma Análise do Código Fonte

Diante do cenário de uma estrutura com muitas incidências de ciclos diretos entre as classes, surgiu então a necessidade de verificar diretamente no código fonte como acontecia esse fenômeno. O resultado dessa investigação foi a identificação de um padrão que se repetia em quase todas as classes com esse comportamento.

A título de exemplificação deste comportamento, as Figuras 29 e 30 mostram um trecho de código das classes *TaskGroup* e *TaskGroupProgression*, respectivamente. Estas classes demonstram serem responsáveis pelo aspecto de tarefas em grupo a serem desempenhadas pelos jogadores no sistema. Os trechos destacados mostram como se deu esta dependência cíclica direta entre as duas classes.

Fonte: Próprio autor

```
public class TaskGroup {

    @JsonIgnore
    @OneToMany(mappedBy = "taskGroup")
    private List<TaskGroupProgression> progressions;
```

Figura 29 – Trecho de Código da Entidade *TaskGroup*

Na classe *TaskGroup* podemos observar o mapeamento do atributo privado *progressions*, que representa uma lista de *TaskGroupProgression*. O mapeamento *@OneToMany* indica que um *TaskGroup* pode ser composto por muitos *TaskGroupProgression* (HIBERNATE, 2009). Este mapeamento fará com que a entidade *TaskGroupProgression* possua uma referência à entidade *TaskGroup*, quando as tabelas forem criadas no banco de dados.

Fonte: Próprio autor

```
public class TaskGroupProgression extends Progression {

    @JsonIgnore
    @ManyToOne
    @JoinColumn(name = "task_group_id", nullable=false)
    private TaskGroup taskGroup;
```

Figura 30 – Trecho de Código da Entidade *TaskGroupProgression*

Já na classe *TaskGroupProgression*, também é possível perceber o mapeamento de uma propriedade privada chamada *taskGroup*, indicando que uma *TaskGroup* pode ter vários *TaskGroupProgression*. A anotação *@ManyToOne* descreve esse tipo de relacionamento (HIBERNATE, 2009). Somente em descrever este relacionamento, já é possível notar uma certa redundância existente no código destas classes.

Este mapeamento bidirecional entre as classes, os quais representam entidades do banco de dados, se mostrou ser muito comum entre as classes com dependência cíclica direta. Do ponto de vista da tecnologia *Hibernate*, o mapeamento bidirecional é possível, e não seria considerado errado utilizar-se desse artifício (HIBERNATE, 2009). Pelo contrário, a própria documentação comenta que o mapeamento bidirecional é mais eficiente na hora de gerenciar o estado de persistência de uma coleção, quando comparado ao mapeamento unidirecional (HIBERNATE, 2009). De fato, somente com o código fonte, não há como afirmar com toda a certeza se essa era a intenção dos desenvolvedores ao incluirem tantos mapeamentos bidirecionais no sistema, durante a gamificação.

Uma das alternativas para chegar a conclusão de que estes mapeamentos foram intencionais é a efetuação da correlação entre a implementação do código e as declarações dadas pelos desenvolvedores, especialmente aquelas referentes às constantes mudanças na modelagem do banco de dados para otimizar o gerenciamento das regras e elementos do jogo.

Entretanto, existem algumas desvantagens em utilizar esse tipo de mapeamento. Uma delas recai no que já vimos em relação aos níveis de acoplamento do sistema, o que pode facilitar o surgimento de alguns dos sintomas descritos por (MARTIN, 2002), tais como *Rigidez*, por causa da dificuldade em fazer alterações que não exijam outras alterações; *Fragilidade*, pois as mudanças no sistema podem acabar fazendo este quebrar em partes conceitualmente não relacionadas; *Imobilidade*, pois é difícil fazer um sistema com altos níveis de acoplamento ser quebrado em partes reusáveis.

Outra desvantagem está ligada a perda de controle na recuperação de uma quantidade específica de itens do banco, impossibilitando um recurso de paginação, por exemplo. Se utilizarmos o exemplo da entidade *TaskGroup* da Figura 29, o modo que a propriedade *progressions* foi mapeada não permite a limitação na quantidade de itens que podem ser recuperados do banco por vez, se a busca for feita a partir desta entidade.

Alguns praticantes afirmam que a melhor forma de utilizar mapeamentos bidirecionais se dá quando o número de filhos daquele mapeamento é limitado e quando o relacionamento entre as entidades é forte. [1] Utilizando o exemplo de uma aplicação que representa uma prova de questões de múltipla escolha. Ao serem criadas as perguntas, estas são acompanhadas de suas respectivas alternativas e a intenção é fazer com que quando uma pergunta for consultada, as alternativas também sejam fornecidas, ou seja, essas duas entidades têm um relacionamento muito forte entre si e ainda o número de alternativas

[1] https://www.callicoder.com/hibernate-spring-boot-jpa-one-to-many-mapping-example/

para cada pergunta é limitado.

Portanto, seria necessária uma análise do ponto de vista da utilidade e funcionalidade de cada uma dessas entidades envolvidas em dependências cíclicas diretas, para investigar se estas possuem, de fato, relacionamentos com estas características descritas acima. Não sendo assim, seria o caso de eliminar as dependências desnecessárias, a fim de se obter níveis de acoplamento mais baixos no sistema, melhorando assim a qualidade da arquitetura.

4.3.2 Grupo 04 - Implementar o Sistema Gamificado

Neste grupo estão presentes as atividades desafiadoras relativas a implementação do VazaZika. Os grandes desafios mencionados pelos desenvolvedores dizem respeito a preocupação com a grande quantidade de eventos dos usuários no novo sistema, devido a constante computação dos elementos do jogo, tais como pontos, medalhas e rankings. Esse fator elevaria consideravelmente o número de interações entre o lado cliente da aplicação e o lado do servidor.

Isso se tornou ainda mais preocupante, segundo eles, devido as limitações de escalabilidade do servidor disponível, para suportar uma quantidade elevada de eventos. Por essa razão, o time de desenvolvimento gastou muitos esforços para tentar diminuir ao máximo essa quantidade de interações entre o cliente e o servidor.

Um dos fatores técnicos que facilitou essa percepção por parte dos desenvolvedores está relacionado a necessidade de sincronização dos dados dos jogadores nos dispositivos. A constante checagem de regras e recompensas do jogo foi um obstáculo considerável a ser superado.

Diante desses desafios, podemos considerar que um dos objetivos primários da equipe de desenvolvimento foi não somente a diminuição das interações entre cliente e servidor, como também a eliminação de qualquer tráfego de informações desnecessárias nestas interações, devido aos recursos de infraestrutura limitados.

Entretanto, um olhar sobre alguns fluxos da gamificação no código chamou atenção para algumas propriedades que aparentavam não estar sendo utilizadas. É o caso do trecho de código obtido a partir da classe *TaskGroupService* mostrado na Figura 31.

Neste trecho é apresentado o método *trackProgress*, o qual recebe um parâmetro representando uma tarefa realizada por um jogador. Resumidamente, o método computa a progressão da tarefa que o jogador desempenhou no sistema. Mas o ponto de curiosidade observado aqui, diz respeito ao fluxo de processamento desta progressão, no qual, em nenhum momento é observada a necessidade de haver uma propriedade mapeada na classe *TaskGroup*, o qual representa uma lista de *TaskGroupProgression* visto na Figura 29, uma vez que todos os cálculos são realizados tendo como base o mapeamento apenas do lado da classe *TaskGroupProgression*.

Fonte: Próprio autor

```
    public void trackProgress(PerformedTask task) {
        Set<TaskGroup> affectedGroups = new HashSet<>();
        Player player = task.getPlayer();
        List<TaskAssignment> incompleteAssignments = taskAssignmentDao.findAllIncomplete(task);
        for (TaskAssignment assignment : incompleteAssignments) {
            if (!assignment.getTaskGroup().canProgress(player)) {
                continue;
            }

            /*
             * Computes and updates the progress in the assignment
             */
            TaskAssignmentProgression progression = this.getAssignmentProgression(assignment, player);
            int completedWorkBefore = progression.getCompletedWork();
            int completedWork = taService.completedWork(task, assignment);
            if (completedWorkBefore != completedWork) {
                int workload = assignment.getWorkload();
                progression.computeProgress(completedWork, workload);
                taProgressionDao.save(progression);
                affectedGroups.add(assignment.getTaskGroup());
            }
        }
        trackGroupsProgress(affectedGroups, player);
    }
}
```

Figura 31 – Trecho de Código da Classe *TaskGroupService*

A Figura 32 mostra os outros métodos da classe *TaskGroupService* que são chamados a partir do método apresentado na figura anterior. Não há evidências de utilização da propriedade *progressions* demonstrada na Figura 29. As suspeitas foram confirmadas quando descobriu-se que a única referência à propriedade mencionada acima estava presente apenas na classe de testes *TaskGroupProgressionRepositoryTest*.

Fonte: Próprio autor

```
    private TaskGroupProgression getGroupProgression(TaskGroup group, Player player) {
        TaskGroupProgression progression = tgProgressionDao.findByPlayer(group, player);
        if (progression == null) {
            progression = new TaskGroupProgression();
            progression.setPlayer(player);
            progression.setTaskGroup(group);
            progression = tgProgressionDao.save(progression);
        }
        return progression;
    }

    private void trackGroupsProgress(Set<TaskGroup> groups, Player player) {
        for (TaskGroup group : groups) {
            TaskGroupProgression progression = getGroupProgression(group, player);
            boolean isCompleteBefore = progression.isComplete();
            Long completedWork = playerDao.sumCompletedWork(group, player);
            Integer workload = group.getWorkload();
            progression.computeProgress(completedWork.intValue(), workload);
            tgProgressionDao.save(progression);
            if (!isCompleteBefore && progression.isComplete()) {
                // the player just completed the task group
                player.addXp(group.getGivenXp());
            }
        }
    }
```

Figura 32 – Trecho de Código da Classe *TaskGroupService*

Isto sugere um tipo de mapeamento, aparentemente, desnecessário provocando uma erosão na arquitetura, ao gerar um relacionamento cíclico direto. Esta informação nos

leva a pensar no que aconteceria, caso a propriedade de um dos lados do relacionamento bidirecional fosse removida.

4.3.3 Grupo 05 - Entender o Sistema Legado

Este grupo reune as atividades desafiadoras relacionadas as dificuldades de entender a plataforma existente, ou seja, o sistema legado. A maior dificuldade relatada pelos entrevistados diz respeito a falta de documentação apropriada da plataforma VazaDengue. Poucos materiais desse tipo foram elaborados e os que haviam estavam, em grande parte, desatualizados.

Esse de fato era um desafio para que a gamificação fosse possível na plataforma, pois mesmo antes de ser gamificada, o sistema já possuia mais de 6000 linhas de código, como já foi mostrado, e mais de 500 métodos. O objetivo dos desenvolvedores estava focado em encontrar, principalmente, oportunidades de reuso das funcionalidades preexistentes do sistema.

A relação dos desafios deste grupo com os impactos na arquitetura dizem respeito as soluções aplicadas pelos envolvidos no projeto de gamificar o sistema, para superar as dificuldades de entendimento do sistema legado. Foram identificadas duas soluções aplicadas diretamente nos desafios deste grupo, estas são descritas a seguir.

A) Engenharia Reversa. Esta solução foi aplicada para o entendimento do sistema e suas funcionalidades. Os envolvidos no projeto de gamificação começaram a tentar entendê-lo a partir do uso da interface das telas do sistema preexistente.

B) Geração Automática de Documentação de API. Esta solução foi aplicada não exatamente para resolver o problema de falta de documentação do sistema legado, mas para evitar a propagação deste problema no sistema que seria gamificado.

Estas duas soluções são válidas, afinal são formas de evitar problemas futuros, e de tentar entender o sistema existente. Porém, é de se questionar a falta de menção a alguma ferramenta de análise arquitetural. Uma vez que o objetivo seria entender a arquitetura do sistema legado e identificar oportunidades de reuso. A utilização de alguma ferramenta do tipo ajudaria bastante nessa tarefa. O presente estudo é uma prova de que ferramentas de análise arquitetural podem fornecer informações valiosas sobre os sistemas de software.

4.3.4 Grupo 08 - Testar o Sistema Gamificado

Por fim, o grupo 8 traz os desafios referentes aos testes do sistema gamificado. Estes desafios não foram considerados específicos da gamificação, uma vez que as dificuldades mencionadas muito se assemelham às mesmas encontradas nas atividades de testes de um sistema comum, não gamificado.

Um dos desenvolvedores responsáveis pelos testes do sistema gamificado chegou a mencionar nas entrevistas que “não houve diferença em implementar os testes de unidade para o VazaZika e implementar testes de unidade para qualquer outro sistema não gamificado”.

Fonte: Próprio autor

```
public class BadgeRepositoryTest extends DbTestUtil{
    @Autowired
    private BadgeRepository badgeDao;

    @Autowired
    private TaskGroupProgressionRepository tgpDao;

    private TaskGroupProgression groupProgression;

    @Autowired
    private PlayerRepository playerDao;

    private Player bob;

    @Autowired
    private TaskGroupRepository tgDao;

    private TaskGroup taskGroup;
```

Figura 33 – Trecho de Código da Classe *BadgeRepositoryTest*

Entretanto, esse grupo foi selecionado, pois com as investigações, foi possível perceber que o processo de gamificação teve sua influência nos testes de unidade implementados no sistema. Um dos efeitos colaterais da gamificação pode ser encontrado na Figura 33, que mostra um trecho da classe *BadgeRepositoryTest*. Nesta classe, apesar das duas propriedades em destaque serem declaradas, mais uma vez apenas um dos lados do mapeamento é utilizado. A Figura 34 mostra o método *initializer* da mesma classe. A linha destacada indica o momento em que o mapeamento é utilizado numa das direções, no entanto o caminho inverso não é explorado.

Embora não percebido pelos desenvolvedores, ou pelo menos, não mencionado durante as entrevistas que intencionavam coletar os desafios enfrentados por eles no processo de gamificação, houve um esforço adicional, por mínimo que seja, na implementação dos testes unitários, devido a modelagem dos relacionamentos bidirecionais entre algumas classes do sistema. Este impacto deve ser levado em consideração, caso haja um estudo futuro que verifique o custo benefício desses mapeamentos, os quais aparentam ter surgido como uma solução para o problema de infraestrutura limitada já mencionados anteriormente neste estudo.

Fonte: Próprio autor

```
@Before
public void initializer() {
    taskGroup = new TaskGroup();
    taskGroup.setGivenXp(2);
    taskGroup.setBadge(badgeDao.findOne(4L));

    taskGroup = tgDao.save(taskGroup);

    bob = playerDao.findOne(2L);

    groupProgression = new TaskGroupProgression();

    groupProgression.setPlayer(bob);
    groupProgression.setTaskGroup(taskGroup);

    //save the TaskGroupProgression
    groupProgression = tgpDao.save(groupProgression);
}
```

Figura 34 – Método *initializer* da Classe *BadgeRepositoryTest*

4.4 REFATORANDO O VAZAZIKA

Durante as descobertas encontradas na fase de exploração dos relacionamentos entre os desafios enfrentados pelos desenvolvedores e os impactos que a gamificação causou à arquitetura, mais precisamente, durante a análise do grupo 4, onde percebeu-se que um dos lados da relação bidirecional entre duas classes não era explorada, observou-se então uma oportunidade de investigação mais aprofundada, para saber se esse tipo de fenômeno se repetia entre as outras classes.

Como ponto de partida, foi tomado o grupo cíclico que comportava o maior número de classes, o qual foi mostrado anteriormente na Figura 23. Nela podemos observar que a maior parte dessas classes estão diretamente relacionados ao jogo implementado no sistema, conforme mostrado na Seção 4.2.2. Com o objetivo de desfazer os ciclos diretos entre as classes, sem causar erros de compilação no projeto, cada um deles foi avaliado cuidadosamente, a partir da indicação visual fornecida pela ferramenta de análise arquitetural. Em seguida, o mesmo processo foi realizado no outro grupo cíclico menor da Figura 24, a qual mostra o segundo maio grupo de classes envolvidas em ciclos, a fim de observar quais seriam os novos índices de acoplamento do sistema, caso não houvesse esse tipo de relacionamento bidirecional repetidas vezes na arquitetura.

4.4.1 Processo de *Refactoring*

O sistema gamificado foi aberto na IDE Eclipse, na qual o mesmo foi desenvolvido. Cada classe com dependência cíclica direta foi avaliada em ambos os lados. A propriedade de um dos lados que referenciava o outro era comentada no código, juntamente com seus métodos de acesso público, *gets* e *sets*. A Figura 35 mostra o exemplo desse processo sendo realizado na classe *Badge*, cujo trecho de código foi comentado para desfazer o relacionamento bidirecional com a classe *TaskGroup*. A própria IDE ficava encarregada de verificar, de forma rápida, se havia alguma referência àquela propriedade no sistema. Caso houvesse, a ferramenta apontava os erros causados pela remoção da propriedade daquela classe no sistema.

Fonte: Próprio autor

Figura 35 – Alteração da Classe Badge

Se a IDE apontasse diversos erros no sistema, devido à remoção, esta propriedade era readmitida e a tentativa era feita do outro lado do relacionamento. Caso os erros apontados pela IDE só dissessem respeito às classes de testes, essas referências nessas classes também eram comentadas, pois não fazia sentido deixar um relacionamento bidirecional, somente para, aparentemente, atender à referência feita por uma classe de testes. Assim, o processo foi sendo repetido, passando por cada classe até chegar ao último que estivesse envolvido em dependência cíclica direta. A Tabela 5 mostra as classes que foram alteradas nesse processo, entre elas algumas classes de testes também estão incluídas, por causa das referências já mencionadas acima. Ao final, temos um total de 12 classes alteradas durante o refactoring, isso representa apenas 5.5% do total de classes presentes no sistema. Porém, mesmo com poucos

A seção a seguir mostrará os resultados obtidos a partir dos cálculos das métricas de qualidade da arquitetura feitos após a conclusão dessas alterações.

Tabela 5 – Classes Alterados no Refactoring

Badge
Picture
Player
PlayerRepositoryTest
RoleResource
TaskAssignment
TaskGroup
Team
TaskGroupProgressionRepositoryTest
User
UserRole
UserService

Fonte: Próprio autor

4.4.2 Resutados do *Refactoring*

Após as alterações nas classes da Tabela 5, foram comparados os índices de qualidade da arquitetura, ao serem calculadas as métricas de acoplamento e ciclidade novamente. Antes, o maior grupo cíclico era composto por 15 classes, como visto na Figura 23. Após as mudanças, esse grupo diminuiu para 10 classes, conforme mostrado na Figura 36. O outro grupo cíclico, por sua vez, era composto por 12 classes e passou a possuir apenas 4, conforme apresentado na Figura 37.

Estas quebras de relacionamento bidirecional entre as classes ocasionou uma melhoria significativa nos índices de acoplamento da arquitetura. A Figura 38 mostra o novo valor do número de dependências cumulativas (CCD) do sistema após as mudanças realizadas nas classes mencionadas. Antes este número era de 5505 e passou a ser 4366, uma queda de mais de 1100 dependências no sistema em geral, representando uma diminuição de mais de 20% das dependências no sistema. Consequentemente, a média de dependências por classe também diminuiu, passando de 24.8 anteriormente a 19.67 após as mudanças, representando a mesma porcentagem de diminuição da média das dependências. A Figura 39 mostra o novo gráfico gerado a partir desse novo cálculo.

Ambos os gráficos apresentam os valores das métricas calculadas para o sistema antes e depois de ser refatorado. Podemos observar nas figuras o quanto o número de dependências diminuiu após o processo de *refactoring*.

Fonte: Próprio autor

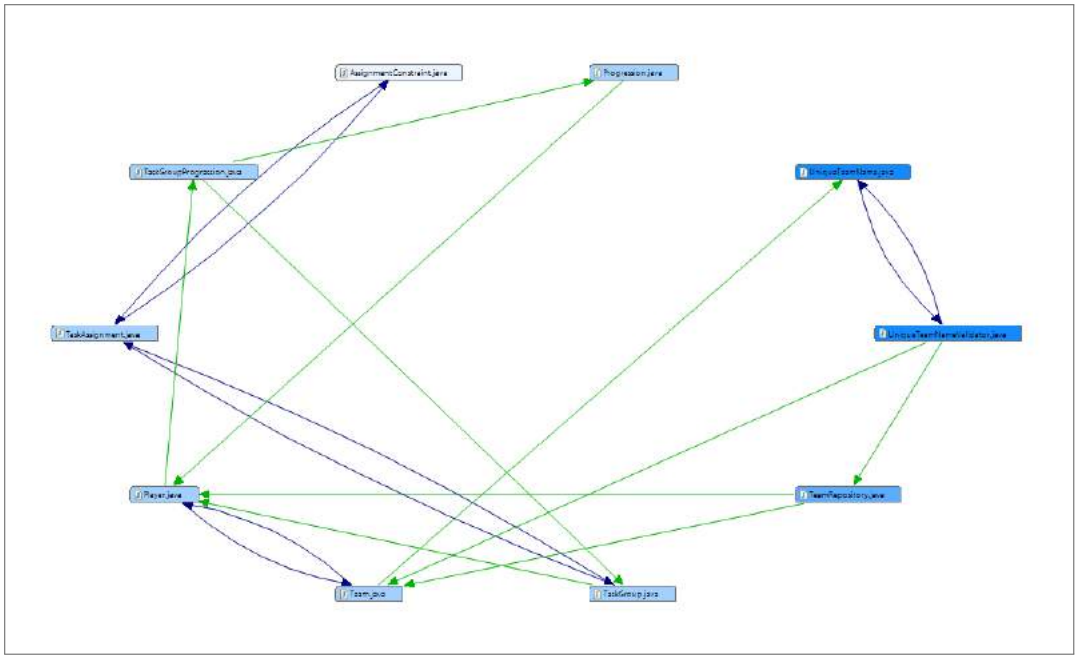

Figura 36 – Grupo Cíclico 1 Após Alterações

Fonte: Próprio autor

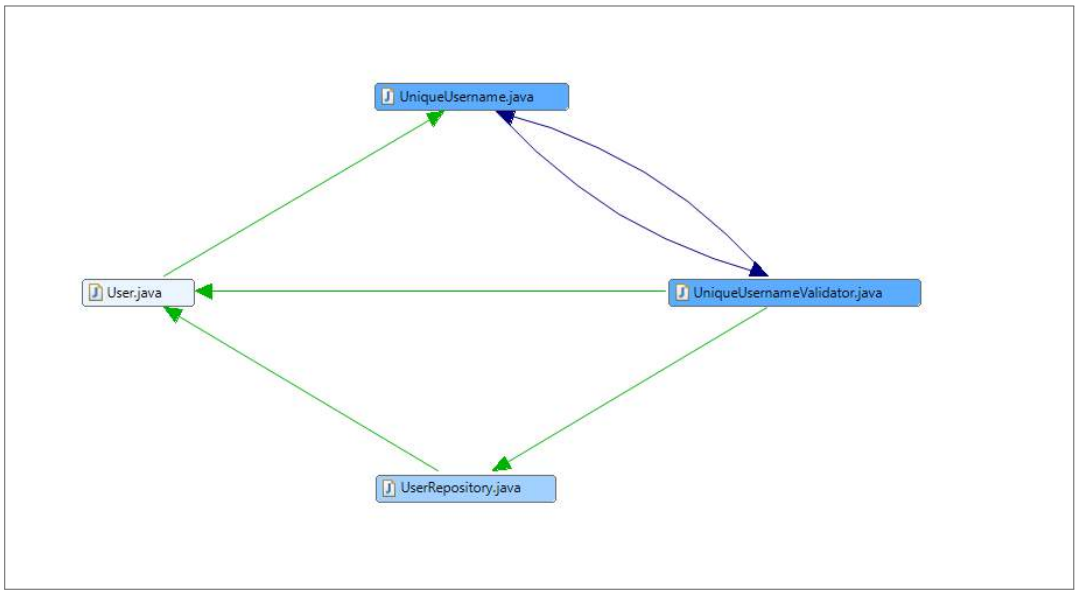

Figura 37 – Grupo Cíclico 2 Após Alterações

Fonte: Próprio autor

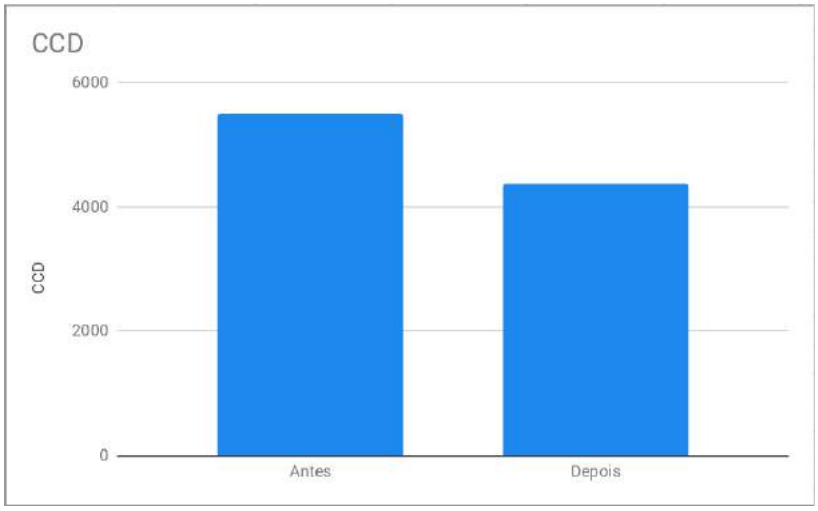

Figura 38 – CCD Antes e Após Refactoring

Fonte: Próprio autor

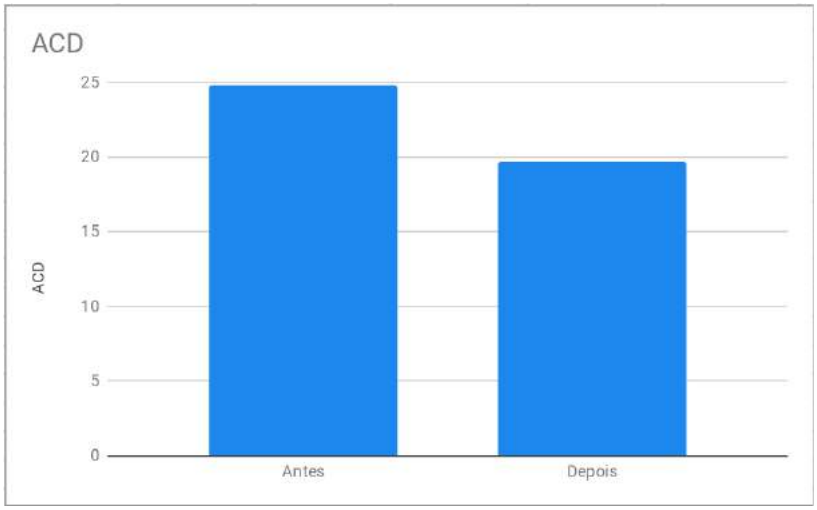

Figura 39 – ACD Antes e Após Refactoring

4.5 CONCLUSÃO DO CAPÍTULO

Neste capítulo vimos o quanto a introdução de gamificação em um sistema existente pode afetar a qualidade da arquitetura deste. Vimos o crescimento do VazaZika em termos de número de linhas de código e de número de classes. Notamos porém, que a diminuição da qualidade da arquitetura não seguiu as mesmas proporções da evolução do tamanho do sistema, sinalizando assim que o processo de gamificação teve forte influência na qualidade da arquitetura final.

Percebemos também que a grande quantidade de classes envolvidas em ciclos fez aumentar os níveis de acoplamento do sistema, tornando-se assim o principal fator de influência do resultado final da qualidade da arquitetura observada. As classes envolvidas em ciclos diretos, mais especificamente, se mostraram verdadeiros vilões da qualidade arquitetural, uma vez que foi descoberto que boa parte dos relacionamentos bidirecionais entre essas classes, aparentemente, não eram necessários. Aprendemos então que esse tipo de artifício deve ser explorado com cuidado, pois a livre utilização pode causar uma erosão estrutural na arquitetura.

Vimos também que praticamente todos os resultados e discussões ora obtidos neste trabalho se deram, graças ao auxílio das ferramentas de análise arquitetural estática. Com a ajuda das ferramentas STAN e SonaGraph, a identificação dos problemas arquiteturais se tornou mais fácil. Com o auxílio dos recursos visuais da ferramenta SonarGraph, por exemplo, os pontos críticos implementados no código foram identificados de maneira mais simples e prática.

A utilização de ferramentas desse tipo se mostrou ser quase que indispensável quando deseja-se implementar gamificação em um sistema existente. Caso esse recurso tivesse sido explorado, provavelmente as decisões relacionadas a gamificação do VazaZika poderiam ter sido tomadas de maneira mais segura, as oportunidades de reuso identificadas mais rapidamente e possíveis problemas arquiteturais preexistentes poderiam ter sido identificados e sanados com mais eficiência. Além disso, as ferramentas podem ajudar a evitar problemas arquiteturais durante o processo de gamificação.

5 CONCLUSÕES E TRABALHOS FUTUROS

Este estudo mostrou uma abordagem da implementação de gamificação no VazaZika, uma plataforma de software, juntamente com os desafios percebidos pelos desenvolvedores e os impactos percebidos na arquitetura a partir da introdução de gamificação no então VazaDengue.

Os resultados sugerem que a introdução de gamificação num sistema já existente não é uma tarefa simples. É necessário, além do esforço de planejamento e implementação, o conhecimento de abordagens e ferramentas que possam auxiliar nesse processo.

Os impactos identificados na arquitetura tem influência direta da abordagem escolhida para gamificar o sistema. A inteção de diminuir as interações entre os lados cliente e servidor, revelada nas entrevistas com os desenvolvedores, os problemas para entender o então VazaDengue, a infraestrutura limitada são exemplos de fatores decisivos que podem ter gerado os efeitos negativos na arquitetura percebidos na evolução do sistema, ao longo do processo de gamificação.

O estudo revelou também que a utilização de uma ferramenta de análise arquitetural pode ser bastante útil e auxiliar no momento de analisar a arquitetura do sistema, o qual deseja-se gamificar, a fim de se obter um entendimento holístico com mais facilidade e agilidade, identificando assim oportunidades de reuso, problemas arquiteturais preexistentes e, consequentemente, ajudar na tomada de decisão quanto a qual seria a melhor abordagem de gamificação a ser adotada para o sistema em questão.

O estudo também mostrou, em termos quantitativos, o quanto uma arquitetura pode sofrer erosão estrutural, por causa da grande quantidade de relacionamentos bidirecionais entre as classes. A construção destes relacionamentos deve ser realizada com cuidado, pois apesar de não ser tratada como um erro de implementação, ou mesmo não ser enquadrada em algum guia de boas práticas de programação, sendo indicada como uma má prática, por exemplo, a livre criação desses tipos de relacionamento pode gerar altos níveis de acoplamento no sistema, fazendo este apresentar os sintomas descritos por Martin (2002).

5.1 LIMITAÇÕES

Apesar desse estudo se mostrar pioneiro na área de investigação de um software que passou a possuir gamificação em suas características e apesar dos resultados se mostrarem relevantes para futuros praticantes e pesquisadores, algumas limitações precisam ser levadas em consideração.

Os desafios de gamificação mostrados no estudo foram percebidos por um grupo de desenvolvedores com pouca ou nenhuma experiência em gamificação, logo, os resultados obtidos podem ser muito diferentes ao serem comparados com os resultados de um outro

estudo semelhante a depender do nível de experiência dos desenvolvedores porventura entrevistados.

O estudo feito neste trabalho se deu com a utilização de uma única amostra de sistema. O ideal seria comparar o mesmo método aplicado aqui em outros sistemas que não foram projetados, a princípio para comportar requisitos de gamificação, mas que introduziram estas características posteriormente. Com isto, a possibilidade de generalização deste estudo se torna bastante restrita, devido à dificuldade de encontrarmos outros sistemas com essas mesmas características, para fins de comparação.

As mudanças efetuadas no sistema, com o intuito de melhorar a qualidade da arquitetura final foram feitas pelo próprio autor deste estudo, o qual não possuia conhecimento aprofundado das regras e lógicas implementadas no sistema gamificado. Apesar do cuidado em não causar erros de compilação ou mesmo de não alterar qualquer procedimento que denotasse mudança do comportamento esperado do sistema, não há como garantir com toda certeza que as alterações não geraram efeitos colaterais, pois o sistema não foi colocado em execução, após as mudanças, a fim de ser testado.

Outra limitação a ser considerada diz respeito a natureza do sistema sob estudo. Este compreende aspectos de um sistema com uma arquitetura orientada a serviços, o que limita, de certa forma, a comparação dos processos e resultados da gamificação do VazaZika com qualquer outro tipo de software que venha a ser gamificado.

5.2 TRABALHOS RELACIONADOS

Eick et al. (2001) apresentou um estudo de um sistema de software de telecomunicação desenvolvido em C/C++ com 15 anos de existência. Eles derivaram um conjunto de índices a partir dos dados das requisições de mudança do sistema, para medir a extensão da erosão arquitetural e seu impacto no sistema. O estudo mostra um claro relacionamento entre a erosão e o crescente esforço de implementar mudanças. O presente trabalho reforça a descoberta desse relacionamento, no entanto, a intenção do trabalho é mostrar de maneira mais específica a erosão arquitetural sofrida por um sistema existente, a partir da introdução do processo de gamificação neste. Para isto, o trabalho utilizou-se de ferramentas de análise arquitetural estática, a fim de obter os resultados e prover reparação da erosão sofrida pelo sistema.

Godfrey e Lee (2000) descrevem suas análises das arquiteturas extraídas do navegador Web Mozilla (o qual subsequentemente evoluiu para o Firefox) e do editor de texto VIM. Ambos produtos de software mostraram um grande número de interdependências indesejadas entre seus subsistemas núcleo. Na verdade, a arquitetura erodida do Mozilla causou significantes atrasos na entrega do produto e forçou os desenvolvedores a reescreverem alguns módulos praticamente do zero (GURP; BOSCH, 2002). O presente trabalho também mostrou interdependências indesejadas no sistema VazaZika, porém essas interdependên-

cias se mostraram surgir a partir de um processo específico de gamificação, deixando lições aprendidas para futuros pesquisadores e praticantes da área de gamificação.

Sutton (2008) usou a ferramenta Structure101 para fazer algumas análises na evolução do código base do findbugs. As descobertas do estudo o levaram a concluir que a arquitetura do sistema sofreu muita erosão ao longo do tempo. Neste trabalho, porém, as ferramentas utilizadas foram STAN e SonarGraph, esta última recebendo mais destaque, por causa das vantagens já mencionadas neste trabalho. Mais uma vez este trabalho se difere, por se tratar da análise de um processo de gamificação sendo inserido num sistema existente. As métricas para identificar a erosão foram calculadas sob o olhar específico do contexto de gamificação.

Dalgarno (2009) apresenta alguns conceitos dentro do contexto de erosão arquitetural, mostrando que uma arquitetura normalmente, à medida que o sistema evolui, tem sua qualidade reduzida. Um dos aspectos de qualidade que podem ser medidos, de acordo com ele, está ligado a habilidade que a arquitetura de um sistema de software tem de atender aos requisitos dos interessados no sistema. Neste trabalho, porém, os aspectos de qualidade foram medidos sob o olhar das métricas de acoplamento e ciclicidade, as quais estão ligadas também aos aspectos de coesão do sistema. À medida que os números de acoplamento e ciclicidade se elevavam, a conclusão tirada desse fenômeno era de que o sistema tinha diminuído sua qualidade arquitetural.

Merkle (2010) fez um estudo de caso com a arquitetura do eclipse, a fim de descobrir uma possível erosão arquitetural durante a evolução da IDE. Para realizar o estudo, ele também fez uma análise de ferramentas de análise arquitetural estática. Dentre as ferramentas analisadas, a escolhida para realizar o estudo foi a SotoGraph, a qual pertence a memsa empresa que criou o SonarGraph, ferramenta usada neste trabalho. Ele mostrou que com a ferramenta é possível simular refactorings, procedimento este que pode dar maior segurança na hora de efetivar uma mudança nesse sentido, diretamente no código. No entanto, o estudo de Merkle (2010) não chegou a implementar mudanças no código do eclipse, algo que diferentemente foi feito neste trabalho. E mais uma vez, a erosão constatada por Merkle se deu através do processo de evolução normal do sistema sob estudo, diferenciando-se então do presente estudo, devido à natureza de investigação relacionada ao impacto causado pela introdução de gamificação no sistema estudado.

Em relação a gamificação sendo aplicada no contexto da saúde, Garett e Young (2018) identificou na literatura 20 estudos neste contexto. Segundo eles, 60% dos estudos diziam respeito a gamificação sendo aplicada para educar pacientes em relação a certos problemas de saúde, como por exemplo saúde mental, doenças sexualmente transmissíveis, nutrição/atividade física, distúrbios do neurodesenvolvimento, ou outros problemas crônicos de saúde. Os outros 40% estavam relacionados ao uso de gamificação para estimular o aprendizado em novos estudantes de medicina ou educação em ciências da saúde em nível de pós-graduação. Neste trabalho no entanto, foi identificado que a gamificação na

área da saúde foi utilizada com o propósito de engajar os cidadãos a colaborarem com os agentes de saúde na identificação e denúncias de pontos de focos do mosquito da dengue. Portanto o trabalho mostrou uma forma diferenciada de utilizar gamificação no contexto da saúde em contraste com os estudos identificados por Garett e Young (2018).

5.3 TRABALHOS FUTUROS

Diante das conclusões obtidas a partir desse estudo, e com as limitações percebidas durante a pesquisa, algumas oportunidades de trabalhos futuros foram identificadas, como forma de dar continuidade ao trabalho realizado e assim produzir resultados ainda mais relevantes:

- Observar de forma prática quais os custos benefícios da criação de relacionamentos bidirecionais entre classes, em termos de ganho de desempenho em detrimento da qualidade da arquitetura, por conta do aumento dos níveis de acoplamento.
- Fornecer uma alternativa de introduzir gamificação no sistema tratado nesse estudo, demonstrando os benefícios que uma abordagem diferente poderia trazer.
- Comparar os impactos causados na arquitetura do sistema ora tratado aqui com outro sistema atualmente em uso que porventura vier a ser gamificado.
- Observar o desenvolvimento das características de gamificação no sistema Vaza-Zika feito por duas equipes de desenvolvedores diferentes, e comparar os diferentes resultados.

REFERÊNCIAS

AL-DUBAI, S.; GANASEGERAN, K.; ALWAN, M. R.; ALSHAGGA, M. A.; SAIF-ALI, R. Factors affecting dengue fever knowledge, attitudes and practices among selected urban, semi-urban and rural communities in malaysia. *Southeast Asian J Trop Med Public Health*, v. 44, n. 1, p. 37–49, 2013.

ALVES, F. *Gamification: Como criar experiências de aprendizagem engajadoras*. [S.l.]: DVS Editora, 2015.

BARGEN, T. V.; ZIENTZ, C.; HAUX, R. Gamification for mhealth–a review of playful mobile healthcare. *Integrating Information Technology and Management for Quality of Care*, IOS Press, v. 202, p. 225, 2014.

BOND, J. G. *Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C*. [S.l.]: Addison-Wesley Professional, 2014.

BOWEN, T. P.; POST, J. V.; TSAI, J. Software quality measurement for distributed systems. *Techn. rep. RADS-TR/Rome air development center. USA*, Sl, 1983.

BURKE, B. *Gamificar: como a gamificação motiva as pessoas a fazerem coisas extraordinárias*. [S.l.]: DVS Editora, 2015.

CORBIN, J.; STRAUSS, A. et al. Basics of qualitative research: Techniques and procedures for developing grounded theory. Thousand Oaks, CA: Sage, 2008.

CZARNECKI, K.; VALENTE, M. T.; TERRA, R.; BIGONHA, R. S. Recommending refactorings to reverse software architecture erosion. In: *2012 16th European Conference on Software Maintenance and Reengineering(CSMR)*. [s.n.], 2012. v. 00, p. 335–340. ISSN 1534-5351. Disponível em: <doi.ieeecomputersociety.org/10.1109/CSMR.2012.40>.

DALGARNO, M. When good architecture goes bad. *Methods and Tools*, v. 17, n. 1, p. 27–34, 2009.

DETERDING, S.; DIXON, D.; KHALED, R.; NACKE, L. 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.; SICART, M.; NACKE, L.; O'HARA, K.; DIXON, D. Gamification. using game-design elements in non-gaming contexts. In: *CHI '11 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2011. (CHI EA '11), p. 2425–2428. ISBN 978-1-4503-0268-5. Disponível em: <http://doi.acm.org/10.1145/1979742.1979575>.

DHAMA, H. Quantitative models of cohesion and coupling in software. *Journal of Systems and Software*, Elsevier, v. 29, n. 1, p. 65–74, 1995.

EASTERBROOK, S.; SINGER, J.; STOREY, M.-A.; DAMIAN, D. Selecting empirical methods for software engineering research. In: *Guide to advanced empirical software engineering*. [S.l.]: Springer, 2008. p. 285–311.

EICK, S. G.; GRAVES, T. L.; KARR, A. F.; MARRON, J. S.; MOCKUS, A. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, v. 27, n. 1, p. 1–12, 2001.

ERL, T. *Service-oriented architecture*. [S.l.]: Pearson India, 2005. v. 8.

FONSECA, J. J. S. Metodologia da pesquisa científica. 2002.

GARETT, R.; YOUNG, S. D. Health care gamification: A study of game mechanics and elements. *Technology, Knowledge and Learning*, Jan 2018. ISSN 2211-1670. Disponível em: <https://doi.org/10.1007/s10758-018-9353-4>.

GIL, A. C. *Métodos e técnicas de pesquisa social*. [S.l.]: 6. ed. Ediitora Atlas SA, 2008.

GILL, H. K.; GILL, N.; YOUNG, S. D. Online technologies for health information and education: a literature review. *Journal of consumer health on the Internet*, Taylor & Francis, v. 17, n. 2, p. 139–150, 2013.

GODFREY, M. W.; LEE, E. H. Secrets from the monster: Extracting mozilla's software architecture. In: CITESEER. *Proceedings of Second Symposium on Constructing Software Engineering Tools (CoSET'00)*. [S.l.], 2000.

GRAAFLAND, M.; SCHRAAGEN, J. M.; SCHIJVEN, M. P. Systematic review of serious games for medical education and surgical skills training. *British journal of surgery*, Wiley Online Library, v. 99, n. 10, p. 1322–1330, 2012.

GUIMARÃES, S. E. Motivação intrínseca, extrínseca e o uso de recompensas em sala de aula. *A motivação do aluno: contribuições da psicologia contemporânea*, v. 3, p. 37–57, 2001.

GURP, J. V.; BOSCH, J. Design erosion: problems and causes. *Journal of systems and software*, Elsevier, v. 61, n. 2, p. 105–119, 2002.

HEILBRUNN, B.; HERZIG, P.; SCHILL, A. Towards gamification analytics-requirements for monitoring and adapting gamification designs. In: *GI-Jahrestagung*. [S.l.: s.n.], 2014. p. 333–344.

HIBERNATE, M. *Hibernate Reference Documentation*. [S.l.]: Version, 2009.

HOFMEISTER, C.; NORD, R.; SONI, D. *Applied software architecture*. [S.l.]: Addison-Wesley Professional, 2000.

HUNICKE, R.; LEBLANC, M.; ZUBEK, R. Mda: A formal approach to game design and game research. In: *Proceedings of the AAAI Workshop on Challenges in Game AI*. [S.l.: s.n.], 2004. v. 4, n. 1, p. 1722.

ISO/IEC. Information technology - software product quality. part 1: Quality model. In: IEEE. *ISO/IEC: FCD 9126-1.2*. [S.l.], 1998.

JAKTMAN, C. B.; LEANEY, J.; LIU, M. Structural analysis of the software architecture — a maintenance assessment case study. In: _____. *Software Architecture: TC2 First Working IFIP Conference on Software Architecture (WICSA1) 22–24 February 1999, San Antonio, Texas, USA*. Boston, MA: Springer US, 1999. p. 455–470. ISBN 978-0-387-35563-4. Disponível em: <https://doi.org/10.1007/978-0-387-35563-4_26>.

JONES, B. A.; MADDEN, G. J.; WENGREEN, H. J. The fit game: preliminary evaluation of a gamification approach to increasing fruit and vegetable consumption in school. *Preventive medicine*, Elsevier, v. 68, p. 76–79, 2014.

KAPP, K. M. *The gamification of learning and instruction: game-based methods and strategies for training and education*. [S.l.]: John Wiley & Sons, 2012.

KARDAN, A. A.; ARANI, A. K. A novel gamification-based architecture for web environments. In: IEEE. *Web Research (ICWR), 2016 Second International Conference on*. [S.l.], 2016. p. 125–130.

KNODEL, J.; POPESCU, D. A comparison of static architecture compliance checking approaches. In: IEEE. *Software Architecture, 2007. WICSA'07. The Working IEEE/IFIP Conference on*. [S.l.], 2007. p. 12–12.

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

LAKOS, J. Large-scale c++ software design. *Reading, MA*, v. 173, p. 217–271, 1996.

LOSAVIO, F.; CHIRINOS, L.; LÉVY, N.; RAMDANE-CHERIF, A. Quality characteristics for software architecture. *Journal of object Technology*, v. 2, n. 2, p. 133–150, 2003.

MADEIRA, N. G.; MACHARELLI, C. A.; PEDRAS, J. F.; DELFINO, M. C. Education in primary school as a strategy to control dengue. *Revista da Sociedade Brasileira de Medicina Tropical*, SciELO Brasil, v. 35, n. 3, p. 221–226, 2002.

MALONE, T. W. What makes things fun to learn? heuristics for designing instructional computer games. In: ACM. *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*. [S.l.], 1980. p. 162–169.

MALONE, T. W. Heuristics for designing enjoyable user interfaces: Lessons from computer games. In: ACM. *Proceedings of the 1982 conference on Human factors in computing systems*. [S.l.], 1982. p. 63–68.

MARINS, D. R. *Um processo de gamificação baseado na teoria da autodeterminação*. Tese (Doutorado) — Tese de Doutoramento. Universidade Federal do Rio de Janeiro. Rio de Janeiro . . . , 2013.

MARTIN, R. C. *Agile software development: principles, patterns, and practices*. [S.l.]: Prentice Hall, 2002.

MERKLE, B. Stop the software architecture erosion: Building better software systems. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. New York, NY, USA: ACM, 2010. (OOPSLA '10), p. 129–138. ISBN 978-1-4503-0240-1. Disponível em: <http://doi.acm.org/10.1145/1869542.1869563>.

MONK, A.; HASSENZAHL, M.; BLYTHE, M.; REED, D. Funology: designing enjoyment. In: ACM. *CHI'02 extended abstracts on human factors in computing systems*. [S.l.], 2002. p. 924–925.

NGUYEN, V.; DEEDS-RUBIN, S.; TAN, T.; BOEHM, B. A sloc counting standard. In: CITESEER. *Cocomo ii forum*. [S.l.], 2007. v. 2007, p. 1–16.

PASSOS, L.; TERRA, R.; VALENTE, M. T.; DINIZ, R.; MENDONÇA, N. C. Static architecture-conformance checking: An illustrative overview. *IEEE software*, Institute of Electrical and Electronics Engineers, Inc., 345 E. 47 th St. NY . . . , v. 27, n. 5, p. 82–89, 2010.

PEA, R. D. User centered system design: New perspectives on human-computer interaction. 1986.

PEREIRA, P.; DUARTE, E.; REBELO, F.; NORIEGA, P. A review of gamification for health-related contexts. In: MARCUS, A. (Ed.). *Design, User Experience, and Usability. User Experience Design for Diverse Interaction Platforms and Environments*. Cham: Springer International Publishing, 2014. p. 742–753. ISBN 978-3-319-07626-3.

PINK, D. H. *Drive: The surprising truth about what motivates us*. [S.l.]: Penguin, 2011.

RIBEIRO, M. E. M.; PRASNISKI, M. E. T.; GALLON, M. da S.; SANTOS, B. S. dos. Ocorrência de motivação intrínseca e extrínseca na escola. *Revista Thema*, v. 13, n. 2, p. 54–67, 2016.

RUNESON, P.; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, Springer, v. 14, n. 2, p. 131, 2009.

SCHELL, J. *The Art of Game Design: A book of lenses*. [S.l.]: AK Peters/CRC Press, 2014.

SICKKIDS. *Pain Squad app*. 2014. Disponível em: <http://www.sickkids.ca/Research/I-OUCH/Pain-Squad-App/index.html>.

SILVA, L. D.; BALASUBRAMANIAM, D. Controlling software architecture erosion: A survey. *Journal of Systems and Software*, Elsevier, v. 85, n. 1, p. 132–151, 2012.

SONARGRAPH. *Sonargraph Product Family*. 2019. Disponível em: <https://www.hello2morrow.com/products/sonargraph/matrix>.

SONARGRAPH. *Sonargraph User Manual*. 2019. Disponível em: <http://eclipse.hello2morrow.com/doc/standalone/content/>.

SONARSOURCE, S. Sonarqube. *Capturado em: http://www. sonarqube. org*, 2013.

SONI, D.; NORD, R. L.; HOFMEISTER, C. Software architecture in industrial applications. In: *1995 17th International Conference on Software Engineering*. [S.l.: s.n.], 1995. p. 196–196. ISSN 0270-5257.

STAN. *Structure Analysis for Java*. 2019. Disponível em: <http://stan4j.com/>.

SUTTON, I. *Software erosion in pictures – FindBugs*. 2008. Disponível em: <http://www.headwaysoftware.com/blog/2008/11/software-erosion-findbugs/>.

THORSTEINSEN, K.; VITTERSØ, J.; SVENDSEN, G. B. Increasing physical activity efficiently: an experimental pilot study of a website and mobile phone intervention. *International journal of telemedicine and applications*, Hindawi Publishing Corp., v. 2014, p. 8, 2014.

WAINER, J. et al. Métodos de pesquisa quantitativa e qualitativa para a ciência da computação. *Atualização em informática*, Sociedade Brasileira de Computação/Editora PUC Rio Rio de Janeiro, v. 1, p. 221–262, 2007.

WERBACH, K.; HUNTER, D. *For the win: How game thinking can revolutionize your business*. [S.l.]: Wharton Digital Press, 2012.

YIN, R. K. *Estudo de Caso-: Planejamento e Métodos*. [S.l.]: Bookman editora, 2015.