CENTRO UNIVERSITÁRIO DO ESTADO DO PARÁ
ESCOLA DE NEGÓCIOS, INOVAÇÃO E TECNOLOGIA - ARGO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

BRUNO COSTA DE MASI DE AGUIAR

***BUGGY*: SISTEMA DE *BUG TRACKING* PARA PROJETOS.**

BELÉM - PA
2020

BRUNO COSTA DE MASI DE AGUIAR

***BUGGY*: SISTEMA DE *BUG TRACKING* PARA PROJETOS.**

Trabalho de Conclusão de Curso apresentado ao Centro Universitário do Estado do Pará (CESUPA), como parte das exigências para a obtenção do título de bacharel em Ciência da Computação.

Orientadora: MSc. Alessandra Natasha Alcantara Barreiros Baganha

BELÉM - PA

2020

BRUNO COSTA DE MASI DE AGUIAR

***BUGGY*: SISTEMA DE *BUG TRACKING* PARA PROJETOS.**

Trabalho de conclusão de curso apresentado à Escola de Negócios, Tecnologia e Inovação do Centro Universitário do Estado do Pará como requisito para obtenção do título de Bacharel em Ciência da Computação na modalidade PRODUTO.

Data da aprovação: / /

Nota final aluno: _____

Banca examinadora

_____

Prof. MSc. Alessandra Natasha Alcantara Barreiros Baganha

Orientador e Presidente da banca

_____

Prof. MSc. Moshe Dayan Sousa Ribeiro

Examinador interno

_____

Prof. MSc. Ricardo Melo Casseb do Carmo

Examinador interno

Dedico este trabalho aos meus pais Júlio e Luciana, grandes incentivadores. Ao meus tios, principalmente Júlio Domingos, Angela e Gilmar Madureira. À minha avó Roseli. À minha namorada Iasmin por estar sempre presente. À todos meus primos, em especial a Brenda Madureira e aos amigos queridos. E por fim, aos docentes do curso, ao Vitor Hugo pela ajuda durante a matéria de Trabalho de Curso, à banca e à minha orientadora Alessandra Natasha, que sempre teve muita paciência e carinho ao me guiar e compartilhar a sua sabedoria.

RESUMO

Bugs afetam a vida de muitos times de desenvolvimento de software no mundo inteiro e, de modo não incomum, equipes de desenvolvimento desconhecem uma melhor maneira de organizar e resolver essas falhas, de forma que acarretam intermitência, instabilidade e falta de confiabilidade de ambientes inteiros. Este Trabalho de Curso apresenta um sistema que disponibiliza a desenvolvedores de software uma solução para monitorar e identificar *bugs* no código e posterior envio de *report automático* ao sistema primário. O sistema foi desenvolvido a partir de pesquisa bibliográfica e métodos dedutivos com abordagem qualitativa de artigos científicos relacionados ao tema, aplicando também metodologias e práticas de desenvolvimento como *Scrum* e *Domain Driven Design*. Através do uso de questionários, foi possível validar a aplicabilidade do solução no mercado, ressaltando a pertinência de integração com outras plataformas, além da detecção e categorização dos *bugs*.

Palavras-chave: *bugs*; *logger listener*; monitoramento automático de falhas; engenharia de software; metodologia ágil.

ABSTRACT

Bugs affect the lives of many software development teams around the world and, not uncommonly, development teams are unaware of a better way to organize and resolve these flaws, in ways that cause intermittency, instability and unreliability of entire environments. This Final Paper presents a system that provides software developers with a solution to monitor and identify bugs in the code and subsequent sending of reports to the primary system. The system was developed from bibliographic research and deductive methods with a qualitative approach to scientific articles related to the theme, also applying development methodologies and practices such as Scrum and Domain Driven Design. Through the use of questionnaires, it was possible to validate the applicability of the solution in the market, emphasizing the pertinence of integration with other platforms, in addition to the detection and categorization of bugs.

Key-words: *software development*; bugs; software engineering; error monitoring; agile methodology.

LISTA DE ILUSTRAÇÕES

Figura 1: Metodologia *SCRUM* 17

Figura 2: Protótipo da Página inicial (baixo nível) 18

Figura 3: Protótipo da Página inicial (alto nível) 19

Figura 4: exemplo de TDD 20

Figura 5: estrutura de arquivos no *back-end* com DDD 22

Figura 6: Logotipo *Bug*gy 23

Figura 7: Jornada do usuário na plataforma 24

Figura 8: Caso de uso 25

Figura 9: Classes 25

Figura 10: Arquitetura DDD no projeto 26

Figura 11: Listagem de *bugs* em formato Kanban 28

Figura 12: Exemplo de como um erro detectado pelo *Logger Listener* é mostrado para o usuário 29

Figura 13: Precificação da plataforma 30

Figura 14: Formulário para reportar um *bug* 33

Figura 15: Código responsável por ativar o Logger Listener 34

Figura 16: Página de um *bug report* 35

Figura 17: Formulário para pesquisa de mercado 37

Figura 18: Respostas de uso de ferramenta de gerência 38

Figura 19: Respostas de uso de ferramenta de monitoramento 39

SUMÁRIO

1 INTRODUÇÃO **8**

1.1 Problema 10

1.2 Justificativa 11

1.3 Objetivos 11

1.4 Estrutura do trabalho 12

2 REVISÃO BIBLIOGRÁFICA **12**

3 METODOLOGIA **15**

3.1 Desenvolvimento do produto **15**

3.1.1 Mercado e público-alvo 23

3.1.2 Engenharia de software 24

3.1.3 Tecnologias utilizadas 26

3.1.4 Funcionalidades do produto 27

3.1.5 Homologação do MVP 29

3.1.6 Comercialização do produto 30

3.1.7 Produtos correlatos 31

3.2 Área de trabalho/abrangência **33**

3.3 Uso de arquivos de entrada e saída **33**

4 RESULTADOS **34**

5 DISCUSSÃO **36**

6 CONCLUSÃO **40**

7 REFERÊNCIAS **40**

1 INTRODUÇÃO

Estima-se que desenvolvedores de software produzam de 100 a 150 erros a cada mil linhas de código. Mesmo se uma pequena fração - em torno de 10% - desses erros acarretem danos graves, uma aplicação relativamente pequena com 20.000 (vinte mil) linhas, terá em média 200 erros graves de codificação (KRASNER, H. 2018). Entende-se como erros graves, todos os erros que são responsáveis por causar vazamento de dados dos usuários, problemas de funcionalidade, entre outras consequências que podem afetar a reputação de uma empresa entre seus consumidores. A falta de capacidade de gestão de erros pode limitar a operação de um sistema.

A criação de um método para atribuir, dividir e priorizar falhas para correção acaba consumindo muito tempo de uma empresa, podendo também ser ineficaz. O esforço necessário para criar esse método acarreta a necessidade de recorrer à soluções alternativas externas, visto que o gerenciamento de erros é aspecto de alta relevância para o mercado de softwares.

A grande maioria das ferramentas de gerenciamento de erros (*Bug*zilla, 2015; Mantis *Bug* Tracker, 2020) apresentam interfaces com operações simples, sendo necessária a indicação da falha, além da indicação de um desenvolvedor para o seu tratamento.

Existem também alguns estudos que utilizam inteligência artificial para algum parâmetro do rastreamento de *bugs*, como: priorização dos *bugs* (ALENEZI, M.; BANITAAN, S., 2013) ou até mesmo definir o desenvolvedor mais competente para um certo *bug* (NAGUIB, H.; NARAYAN, N.; BRÜGGE, B., 2013). Estas limitações definem uma área inexplorada no gerenciamento de erros, motivando o desenvolvimento de uma ferramenta em formato de subsistema monitor, e a automatização de registro de erros em uma aplicação em tempo real.

Este trabalho objetiva a criação de um sistema denominado *bug tracking*, ou *Buggy* - processo de log de dados e monitoramento de falhas ou erros durante a fase de testes de um software.

Segundo Ahsan, Ferzund e Wotawa(2009) o custo de manutenção de um software pode ser reduzido usando menos recursos, fazendo com que empresas recorram a sistemas de *bug tracking.* Adotar esse processo no desenvolvimento de um software possibilita até mesmo que os usuários finais ajudem as empresas a detectar possíveis *bugs*.

A adoção de uma ferramenta de *bug tracking* auxilia do começo ao fim do desenvolvimento de software, podendo ser a solução de diversos *bugs*. A empresa que adota este tipo de solução se beneficia em relação a entrega de uma aplicação de alta qualidade ao cliente, ajudando a identificar e priorizar problemas, evitando a repetição de tarefas e diminuindo o custo de desenvolvimento.

O *Buggy* pretende ir em direção a diminuição da repetição de tarefas e diminuição do custo de desenvolvimento, a partir de um sub-sistema chamado *logger listener*, que fará com que seja possível a abertura de um *ticket* na plataforma de triagem sempre que um erro aconteça, fazendo com que não seja necessária a etapa de envio manual do erro.

Será também realizada uma pesquisa de campo das aplicações semelhantes que já estão no mercado, incluindo desenvolvedores, visando avaliar qual seria uma abordagem simples, mas eficaz, para lidar com erros de software.

1.1 Problema

No dia em que uma empresa anuncia publicamente uma falha de software, a mesma perde em média US$ 2,3 bilhões dos acionistas (Parasoft, 2015). Uma pesquisa de JONES (2008) apontou que o custo para resolver os

bugs pós-lançamento demanda US$16.000, mas um *bug* encontrado na fase de design orça US$ 25.

Uma pesquisa da CISQ (2018) - empresa de consórcio para informação e qualidade software - concluiu que software de baixa qualidade custou às organizações US $ 2,8 trilhões somente nos EUA, sendo 18,22% desse valor provisionados apenas como dívida técnica. O custo e a probabilidade de *bugs* de software são vultosos. Mas, o impacto financeiro não é o único efeito que a baixa qualidade pode ter em um negócio.

A citada pesquisa aponta ainda que o prejuízo real dos *bugs* de software é triplo. A baixa qualidade afeta clientes, empresa e até mesmo a carreira. As consequências de deixar *bugs* no software precisam ser consideradas.

Nesse contexto, os defeitos podem nem ser críticos, mas podem causar danos significativos à conta bancária e à reputação de uma empresa. Estes aspectos levantados demonstram que o tratamento de erros é fundamental para a manutenção e o bom funcionamento de um sistema, relata a pesquisa. Mas qual seria a melhor abordagem para lidar com os *bugs* de um sistema?

Este trabalho propõe a criação de uma plataforma para desenvolvedores oferecendo a possibilidade de realizar uma triagem de *bugs* nos projetos, e assim dividir e designar a solução desses *bugs* à outros desenvolvedores do time. Inclui-se, ainda, um serviço de monitoramento e identificação de falhas gerando um *report* na plataforma na fase de manutenção/evolução.

1.2 Justificativa

Os *bugs* fazem parte da vida de times de desenvolvimento de software, e caso não sejam resolvidos, podem ser responsáveis pela inviabilidade de um negócio ou resultar no encerramento da plataforma. Por *bugs* pode se entender, para além de uma falha fatal no sistema, quaisquer

erros que façam com que o sistema se comporte de maneira contrária a qual foi projetado.

Este projeto se baseia nos aspectos levantados. O desenvolvimento de um tratamento para filtrar falhas, pode consumir muito tempo em projetos de desenvolvimento de sistemas, podendo ainda não serem eficazes. Além disso, a complexidade de desenvolvimento de um sistema para o tratamento de erros pode levar diversos projetos à recorrer a uma solução direcionada a este tipo de tratamento, visto que é um requisito importante para o desenvolvimento de uma aplicação.

Neste sentido, o *bug tracking* - processo de log de dados e monitoramento de falhas ou erros durante a etapa de teste do software - pode auxiliar na resolução de diversos *bugs*. A adoção de uma solução, neste aspecto, proporciona a entrega de aplicação de maior qualidade; ajuda a identificar, classificar e priorizar falhas; e evita a repetição de tarefas, diminuindo os custos de desenvolvimento de aplicações.

1.3 Objetivos

O objetivo geral do trabalho é desenvolver uma plataforma no qual desenvolvedores possam monitorar, identificar e reportar de forma automática *bugs* de aplicações no ambiente de produção. Os objetivos específicos são:

- Pesquisar e analisar abordagens ou soluções similares;
- Definir o escopo do sistema e requisitos de desenvolvimento;
- Modelar o sistema utilizando princípios de *Scrum*;
- Implementar o sistema respeitando o objetivo geral;
- Documentar o desenvolvimento e os resultados obtidos.

1.4 Estrutura do trabalho

O trabalho consiste em um primeiro capítulo contendo a introdução, onde é apresentado o tema, o problema de pesquisa, a pergunta que norteou o estudo e os objetivos. No segundo capítulo, abordou-se a revisão

bibliográfica, enquanto no terceiro capítulo é apresentada a proposta/metodologia, que se baseou no desenvolvimento da plataforma *Bug*gy juntamente ao *Logger Listener*. O quarto capítulo discorre sobre a análise dos dados, enquanto o quinto, sobre a discussão dos dados; seguido da conclusão.

2 REVISÃO BIBLIOGRÁFICA

Nesta seção é apresentada a revisão bibliográfica necessária para o desenvolvimento deste trabalho, bem como os conceitos fundamentais para se entender as problemáticas que envolvem o sistema.

2.1 Bugs em software

O desenvolvimento de softwares é sujeito à diversos *bugs*/falhas que podem resultar em mau-funcionamento, que pode ameaçar a integridade do sistema, e também a segurança do ambiente computacional. Para corrigir estas falhas, os desenvolvedores enfrentam diferentes desafios, como cronogramas pouco adaptados para a resolução de problemas relacionados a falhas, ou até mesmo a falta de qualificação para lidar com um aspecto tão específico em projeto de desenvolvimento de software (YIN, Zuoning, et al., 2011).

Segundo a IDC, o custo de manutenção de software foi avaliado em 86 bilhões de doláres em 2005, sendo quase ⅔ (dois terços) do custo geral de produção do software. Desenvolvedores gastam 50% a 80% do seu tempo procurando, entendendo e consertando *bugs* nas aplicações. Consertá-los corretamente, e de forma antecipada, irá economizar dinheiro, além de aprimorar a qualidade do software (GU, Z.; BARR, E.T; HAMILTON, D.J; SU, Z., 2010).

2.2 Bug tracking

Segundo Jalberte Weirmer (2008), os sistemas de rastreamento de *bugs* são ferramentas importantes que orientam as atividades de manutenção

dos desenvolvedores de software, a utilidade desses sistemas é dificultada por um número excessivo de relatórios de erros duplicados - em alguns projetos, até um quarto de todos os relatórios são duplicados, o que acontece através de múltiplas fontes reportarem o mesmo erro. Os desenvolvedores devem identificar manualmente os relatórios de erros duplicados, mas esse processo de identificação consome tempo e agrava o já alto custo da manutenção do software.

À medida que os projetos de software se tornam cada vez maiores e complexos, torna-se mais difícil verificar adequadamente o código antes do envio para o ambiente de produção. As atividades de manutenção representam mais de dois terços do custo do ciclo de vida de um sistema de software (BOEHM, B.; BASILI, V.R., 2005), conforme já mencionado, totalizando US $ 70 bilhões por ano nos Estados Unidos (SUTHERLAND, J., 1995).

A manutenção de software é crítica para a confiabilidade do software e os relatórios de defeitos são críticos para a manutenção moderna de software. Muitos projetos de software dependem de relatórios de erros para direcionar a atividade de manutenção correta. Nos projetos de software de código aberto, os relatórios de *bugs* geralmente são enviados por usuários ou desenvolvedores de software e coletados em um banco de dados por uma das várias ferramentas de rastreamento de *bugs*. É permitido que os usuários relatem e potencialmente ajudem a corrigir *bugs* para melhorar a qualidade do software em geral.

Pedidos de mudanças no software são frequentemente geradas durante a evolução de qualquer sistema de software grande. A maioria desses pedidos estão relacionados à manutenção corretiva do software, e estes são chamados de *bug reports* (relatórios de *bugs*). A resolução precisa e oportuna desses relatórios de *bugs* não apenas melhora a qualidade da tarefa de manutenção de software, mas também fornece a base para manter um software específico ativo (AHSAN, S.N.; FERZUND, J.; WOTAWA, F., 2009).

A maioria das grandes empresas de software utilizam um sistema de *bug tracking* para administrar os *bugs* e os desenvolvedores. No desenvolvimento e manutenção de software, um repositório de *bugs* - que o sistema acaba providenciando - é muito significante para armazenar os *bugs* reportados pelos usuários. Esses usuários, incluindo desenvolvedores, testadores e usuários finais, enviam o conteúdo do *bug* como um relatório para identificar defeitos do software ou até mesmo sugestões de usuários (XUAN, J.; JIANG, H.; REN, Z.; YAN, J.; LUO, Z., 2017).

Em muitos projetos, os sistemas de *bug tracking* permitem que os usuários se comuniquem com os desenvolvedores para informar sobre quaisquer *bugs*, oportunizando o desenvolvimento de novas funcionalidades para o software. Todavia, muitos destes sistemas apresentam limitações, e como resultado acabam por exigir em demasia dos usuários finais, que não são familiarizados com práticas de desenvolvimento (JUST, Sascha; PREMRAJ, Rahul; ZIMMERMANN, Thomas., 2008).

2.3 Tecnologias

Neste trecho serão citadas embasamentos teóricos para as duas principais tecnologias que são utilizadas para o desenvolvimento deste produto.

Em pesquisa na página do portal do navegador do Mozilla Firefox (2020), é colocado que o Node.js usa um loop de eventos em vez de threads. Neste é capaz de escalar para milhões de conexões simultâneas. O Node.js aproveita o fato de que os servidores passam a maior parte do tempo aguardando operações de E/S (entrada e saída). Lendo um arquivo de um disco rígido, acessando um serviço da Web externo ou aguardando o término do upload do arquivo, porque essas operações são muito mais lentas do que nas operações de memória. Toda operação de E/S no Node.js são assíncronas.

O servidor continua a processar solicitações recebidas enquanto a operação de E/S está ocorrendo. O JavaScript, que será a principal linguagem usada no projeto, é adequado para programação baseada em

eventos, pois possui funções e fechamentos anônimos que tornam muito fácil definir retornos de chamada em linha, e os desenvolvedores de JavaScript já sabem como programar dessa maneira. Esse modelo baseado em eventos torna o Node.js muito rápido e facilita o dimensionamento de aplicativos em tempo real.

Segundo AGGARWAL (2018), a web moderna está se tornando cada dia mais dinâmica e interativa com o usuário. As tendências de *design* da experiência do usuário estão mudando e evoluindo continuamente. Os *scripts* do cliente agora garantem que apenas os dados necessários e essenciais sejam enviados por push, e uma experiência agradável e contínua seja mantida em toda a Web. É a demanda mundial de hoje por facilidade, eficiência e maior acessibilidade. O ReactJS possui poder e recursos intensos para atender aos requisitos das tendências atuais.

3 METODOLOGIA

A pesquisa se utiliza de pesquisa bibliográfica e métodos dedutivos com abordagem qualitativa de artigos científicos relacionados ao tema, aplicando também metodologias e práticas de desenvolvimento como *Scrum* e *Domain Driven Design*.

3.1 Desenvolvimento do produto

Nesta seção é apresentada como a plataforma foi desenvolvida. Serão mostradas as metodologias de programação utilizadas, além dos passos que foram tomados para idealizar e realizar a implementação do produto.

Uma das metodologias utilizadas no desenvolvimento do projeto se dá pela metodologia ágil *SCRUM*, com o objetivo de fornecer um processo conveniente para o projeto. A *Scrum* apresenta uma abordagem empírica que aplica algumas idéias da teoria de controle de processos industriais para o desenvolvimento de softwares, reintroduzindo as idéias de flexibilidade, adaptabilidade e produtividade. O foco da metodologia é encontrar uma forma

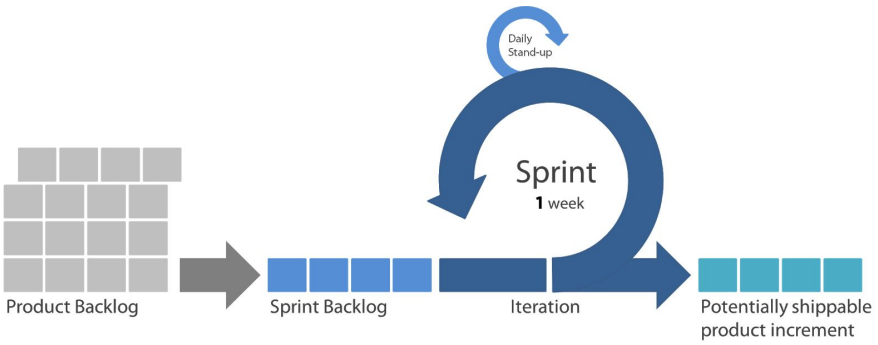
de trabalho dos membros da equipe para produzir o software de forma flexível e em um ambiente em constante mudança.

A idéia principal da *Scrum* é que o desenvolvimento de softwares envolve muitas variáveis técnicas e do ambiente, como requisitos, recursos e tecnologia, que podem mudar durante o processo. Isto torna o processo de desenvolvimento imprevisível e complexo, requerendo flexibilidade para acompanhar as mudanças. O resultado do processo deve ser um software que é realmente útil para o cliente.

A *Scrum* divide o desenvolvimento em iterações (*sprints*) de trinta dias. Equipes pequenas, de até dez pessoas, são formadas por projetistas, programadores, engenheiros e gerentes de qualidade. Estas equipes trabalham em cima de funcionalidades (os requisitos, em outras palavras) definidas no início de cada sprint. A equipe é responsável pelo desenvolvimento desta funcionalidade. Na *Scrum* existem reuniões de acompanhamento diárias. Nessas reuniões, que são preferencialmente de curta duração (aproximadamente quinze minutos), são discutidos pontos como o que foi feito desde a última reunião e o que precisa ser feito até a próxima. As dificuldades encontradas e os fatores de impedimento (bottlenecks) são identificados e resolvidos (DOS SANTOS SOARES, 2004).

Como a *SCRUM* é uma metodologia ela não precisa ser seguida a risca, e por causa do tempo limitado de 6 meses para o projeto ser realizado, o tempo de cada sprint foi reduzido para 1 semana.

Figura 1: Metodologia *SCRUM*



Fonte: Zup, 2019.

As primeiras *sprints* foram as responsáveis por gerar a prototipação do projeto, a qual foi desenvolvida usando o programa Adobe XD. Em linhas gerais, prototipação tem como objetivo principal tornar tangível e visual um projeto, da forma mais rápida e econômica possível. Um bom protótipo pode ser ricamente explicativo, visual e até interativo, possibilitando a execução de testes com o usuário final, membros da própria equipe ou parceiros. Esses testes ajudarão a identificar falhas e erros, e o processo de melhorar, testar, melhorar, testar… fica muito mais produtivo e ágil.

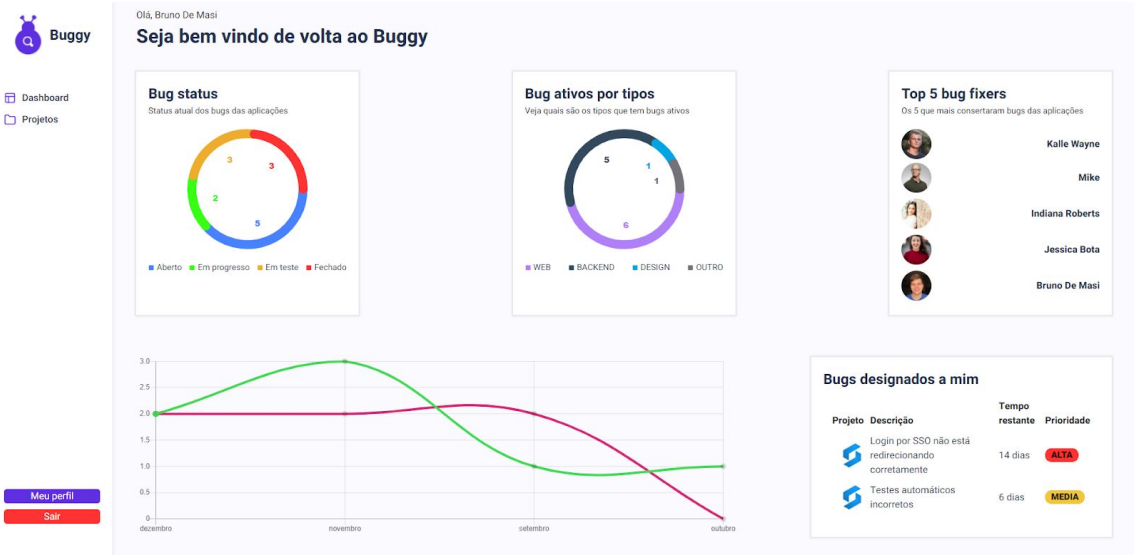
Em relação ao projeto, foi gerado de início um protótipo de baixa fidelidade com as funcionalidades principais do projeto. Com baixa fidelidade tem-se um foco que não é se preocupar com *design*, cores ou disposição dos elementos, mas fazer rápido, com representações superficiais e gerais das funcionalidades e interações com o usuário.

Figura 2: Protótipo da Página inicial (baixo nível)

Fonte : Autor, 2020.

Após chegar em um resultado na qual a Interface do Usuário esteja usável e de acordo com as 10 heurísticas de Nielsen (2005), a próxima etapa foi de transformar o protótipo de baixa fidelidade em um de alta fidelidade adicionando a paleta de cores e o logotipo da plataforma. Tal tipo de protótipo, geralmente é concebido na fase final do projeto de interface e dependendo da ferramenta que estiver utilizando, é possível simular todo o fluxo de informação, navegação e até interação com o usuário.

Figura 3: Protótipo da Página inicial (alto nível)



Fonte : Autor, 2020.

Após validadas as interações com os protótipos, pode-se prosseguir para o desenvolvimento dos códigos para a aplicação, onde foram usadas as tecnologias React e Node - mais sobre na seção 3.1.3 deste artigo-, usando DDD (*Domain Driven Design*) e TDD (*Test Driven Development*) como metodologias e técnicas para o desenvolvimento da aplicação.

O DDD é uma filosofia cujo foco são as complexidades do domínio e onde o objetivo é tornar essas complexidades explícitas no modelo de domínio e na sua implementação no código (LANDRE; WESENBERG; OLMHEIM, 2007), Segundo Eric Evans (2003), autor do livro que deu origem ao DDD, o *design* propõe práticas como:

- **Alinhamento do código com o negócio:** o contato dos desenvolvedores com os especialistas do domínio é algo essencial quando se faz DDD;

- **Favorecer reutilização:** os blocos de construção, facilitam aproveitar um mesmo conceito de domínio ou um mesmo código em vários lugares;
- **Mínimo de acoplamento:** Com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos;
- **Independência da Tecnologia:** DDD não foca em tecnologia, mas sim em entender as regras de negócio e como elas devem estar refletidas no código e no modelo de domínio.

Essas práticas tornam o modelo ótimo para trabalhar em projetos robustos, facilitando a quebra dos componentes da aplicação em módulos onde proporcionará mais facilidades para atender aos complexos processos de negócio.

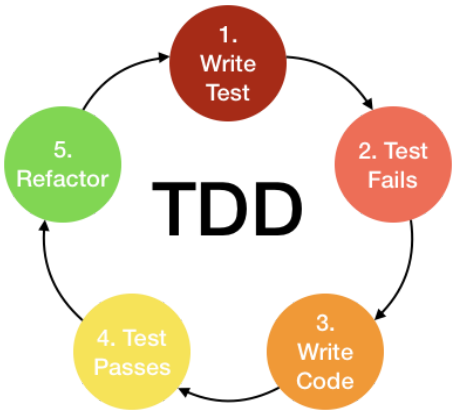
Já o TDD tem uma proposta diferente, a prática começa com pensamentos sobre como testar a funcionalidade necessária. Depois de escrever casos de teste automatizados que geralmente nem compilam, os programadores escrevem código de implementação para passar nesses casos de teste. O trabalho é mantido sob o controle intelectual do programador; já que o programador está continuamente tomando pequenas decisões de implementação e aumentando a funcionalidade em uma taxa relativamente consistente.

Todos os casos de teste que existem para o programa inteiro devem passar com êxito antes que o novo código seja considerado totalmente implementado. Portanto, percebe-se, com certo grau de confiança, que o novo código não introduzirá uma falha ou irá mascarar uma falha na base de código atual. (GEORGE; WILLIAMS, 2004).

Um projeto desenvolvido utilizando TDD e com testes bem construídos resulta em um software de qualidade. Pois, além do código mínimo e

organizado, a técnica proporciona uma documentação, que não apenas condiz com o software, mas também é executável. Quando houver a necessidade de manutenção no software, é possível executar os testes, e assim descobrir o que já está implementado.

Figura 4: exemplo de TDD

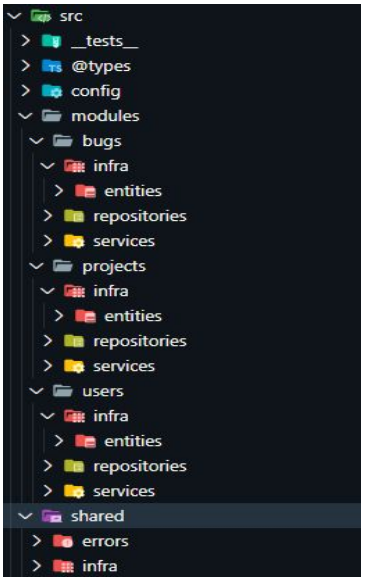


Fonte: Luis Machado, 2018.

Embora TDD e DDD não sejam dependentes um do outro, se têm resultados satisfatórios ao juntá-los em um mesmo projeto. Ambos resultam em um código organizado, além da alta separação de “tarefas” produzidas pelo DDD, devido a metodologia implicar em uma divisão de camadas, e a documentação executável resultante da utilização da técnica TDD. Tal combinação eleva a manutenibilidade do código, garantindo uma maior qualidade do produto final.

Aplicando o DDD no projeto do *back-end* do *Bug*gy, tem-se um resultado como na Figura 5.

Figura 5: estrutura de arquivos no *back-end* com DDD



Fonte: Autor, 2020.

No *front-end* se pode usar uma arquitetura mais simples mas igualmente eficaz para o que propõe, neste caso o DDD não é necessário porém para ter-se uma aplicação com testes constantes para evitar erros inesperados, continua-se usando a técnica TDD.

O resto das *sprints* foram designadas ao desenvolvimento do produto que foi separado em três partes, a primeira sendo dedicada à idealização e realização do *front-end*. Neste tem-se a interface do sistema, desde a navegação entre as páginas e o *design*. Isso contribui para um primeiro ponto de validação, porque é onde o cliente consegue visualizar a plataforma, interagir e identificar quais são os possíveis pontos de melhoria para aprimorar, cada vez mais, a experiência e usabilidade do projeto.

Figura 6: Logotipo *Bug*gy

Fonte: Autor, 2020.

Na segunda parte, foi feito o desenvolvimento do *back-end*, seguindo os princípios e metodologias já apresentados. Na terceira foi desenvolvido o módulo chamado *Logger Listener,* o qual trabalha diretamente no projeto do usuário, monitorando e detectando algum erro que possa vir a acontecer. Ao identificar um erro, o sistema envia a falha detectada para a API do *Bug*gy com uma chave específica do projeto, criando assim uma nova ocorrência de falha junto aos dados capturados da mesma.

3.1.1 Mercado e público-alvo

O projeto tem como público-alvo o mercado de TI (Tecnologia da Informação), que teve um crescimento apontado de 6,7% no setor global em 2019, sendo que no Brasil cresceu 9,8%. São cerca de 19 mil empresas atuando no setor de Software e Serviços no Brasil, sendo delas 5.294 voltadas ao desenvolvimento e produção de software (ABES, 2019).

Para o crescimento global, no ano de 2020, pode ser feito um destaque para a pandemia do COVID-19 (Corona Virus Disease), onde a mesma, apesar das dificuldades, contribuiu para uma projeção de US$131 bilhões para US$295 bilhões de doláres no mercado global de Tecnologia até o ano de 2025 (Market Data Forecast, 2020). A partir dos dados é possível

perceber um crescente número de empresas desenvolvendo software, o que acarreta um crescimento nas quantidades de softwares que serão criados.

Assim, conforme visto anteriormente no item 2.2, todos esses softwares irão precisar de alguma ferramenta que os ajude em relação às falhas. Com esses fatos analisados vem a decisão de atender aos times de desenvolvimento com o produto, tendo o objetivo de ajudar a facilitar o trabalho e o esforço que os times precisam em relação a rastreamento de falhas.

3.1.2 Engenharia de software

A engenharia de software vai muito além da programação e deve-se estar atento aos apelos do contratante, do cliente final e dos usuários do programa desenvolvido (QueroBolsa, 2019). Nesta seção serão apresentados casos pensando na interação do usuário com a plataforma, de forma detalhada.

3.1.2.1 Infográfico

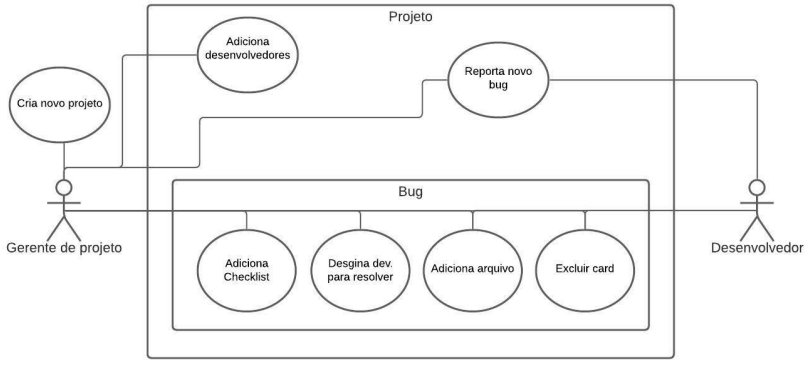
Figura 7: Jornada do usuário na plataforma

Fonte: Autor, 2020.

Com o infográfico é explicado qual o fluxo que o usuário deve seguir na aplicação. Ajuda a auxiliar na compreensão do leitor, enquanto resume os módulos da aplicação para fácil explicação

3.1.2.2 Diagrama de caso de uso

Figura 8: Caso de uso

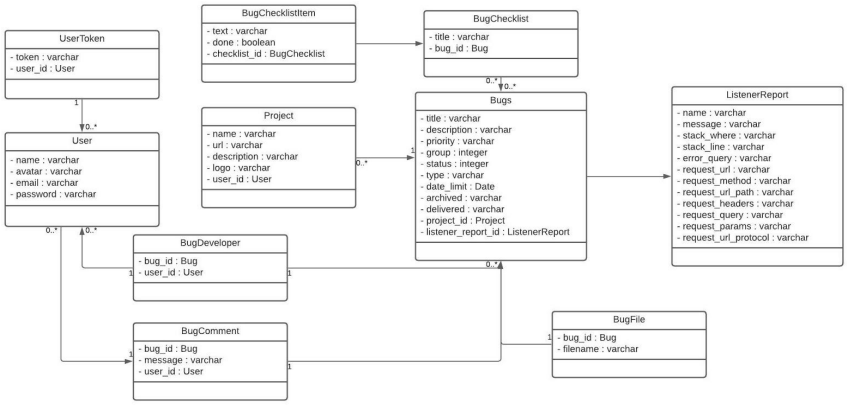


Fonte: Autor, 2020.

Nesse diagrama é possível ver as interações dos usuários com a plataforma. Por gerente de projeto, se define qualquer usuário que vá criar um projeto no *Bug*gy e administrar os membros e os *bugs*.

3.1.2.3 Diagrama de classes

Figura 9: Classes

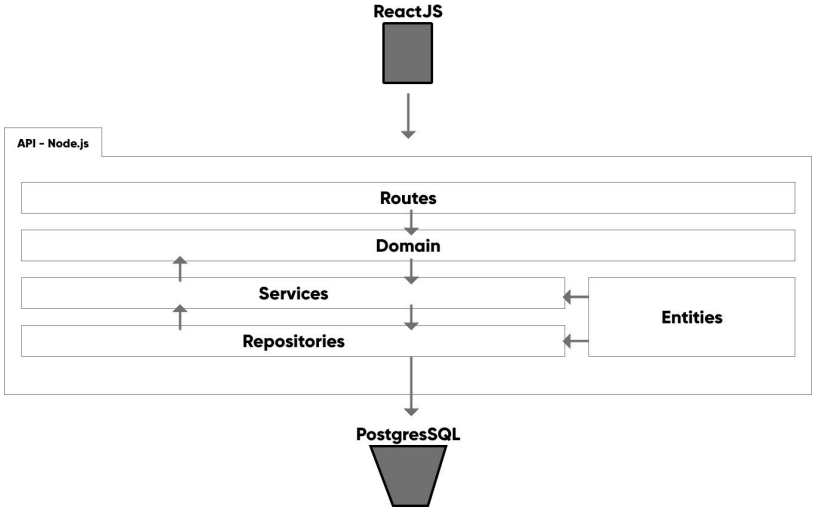


Fonte: Autor, 2020.

Nesta seção se apresenta o diagrama de classes, demonstrando as entidades e os atributos de cada. Onde se vê maior interação com as outras entidades é nos *bugs.* Fato esperado por ser o foco principal da plataforma.

3.1.2.4 Arquitetura e processo de software

Figura 10: Arquitetura DDD no projeto



Fonte: Autor, 2020.

3.1.3 Tecnologias utilizadas

O sistema como um todo será dividido em três partes, sendo uma delas a API (Interface de Programação de Aplicações) central que irá conter toda a lógica de servidor das nossas aplicações. A API será desenvolvida usando a linguagem JavaScript, rodando o Node.js que é um interpretador construído com base da engine V8 do Google, sendo possível fazer um código JavaScript aberto de alta performance, assíncrono e orientado a eventos (TILKOV, S.; VINOSKI, S., 2010).

O Node.js pode ser usado para construir um ambiente de servidor leve e performático, ideal para construir APIs (CHANIOTIS, I. K.; KYRIAKOU, K.-I. D.; TSELIKAS, N. D., 2015). É usado com este mesmo propósito por diversas empresas grandes, como o Walmart (O'DELL, J., 2012).

Todas as requisições serão feitas pelo protocolo HTTP (Hypertext Transfer Protocol), que será gerenciado no Node pela biblioteca Express que

providencia um framework para aplicações web. Será usado o banco de dados Postgres, por ser gratuito, open-source e altamente customizável (DATABASE STAR, 2019) rodando em um contêiner no Docker, já que este facilita a criação e administração de ambientes isolados (DIEDRICH, C., 2015).

A parte visual do sistema será desenvolvida em React, uma biblioteca JavaScript, que é mantida pelo Facebook, e será usada pela sua performance, mantenabilidade e simplicidade (PANDIT, N., 2020). A aplicação será a interface na qual toda a gerência dos *bugs* e time de desenvolvimento estará presente. Outro sub-sistema será implementado usando Node.js que será responsável por apenas enviar os erros interceptados do sistema enquanto este está em produção, assim que o erro é noticiado, é automaticamente salvo no banco de dados.

Para a última tecnologia que será essencial no envio do software para produção, está o NGINX que foi especialmente desenhado para arquiteturas modernas de arquiteturas de *clouding*. A hospedagem será feita no serviço da DigitalOcean.

3.1.4 Funcionalidades do produto

As funcionalidades são o *core* de todo software, é o que define o que o produto é capaz de fazer. Pensar em um produto significa justamente explorar as diferentes formas em que diferentes funcionalidades podem ser combinadas para resolver o mesmo problema. Nesta seção será introduzida as principais funcionalidades do produto.

3.1.4.1 Login/Cadastro

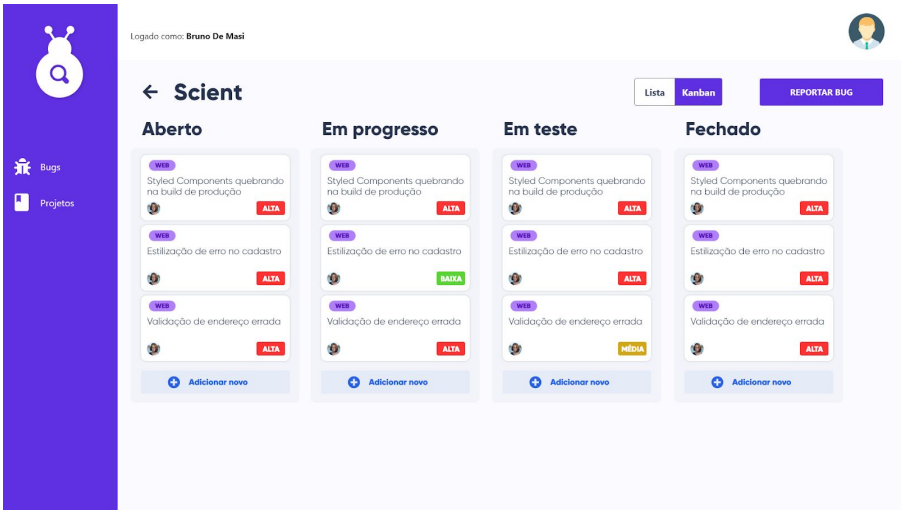
O usuário deve se cadastrar e entrar na plataforma. Entrando dados como Nome, Especialidade, E-mail e Senha, o usuário pode entrar na plataforma e interagir com todas as funcionalidades. Só é possível ter acesso à plataforma com o usuário criado.

3.1.4.2 Projetos

É possível criar projetos onde existirá o gerenciamento dos *bugs* de um dito projeto. Qualquer usuário cadastrado pode criar um novo projeto na plataforma (seguindo as limitações impostas pelos pacotes apresentados na seção 3.1.6). Apenas os donos de projetos podem adicionar mais membros aos seus respectivos projetos.

3.1.4.2 Bugs

Figura 11: Listagem de *bugs* em formato Kanban



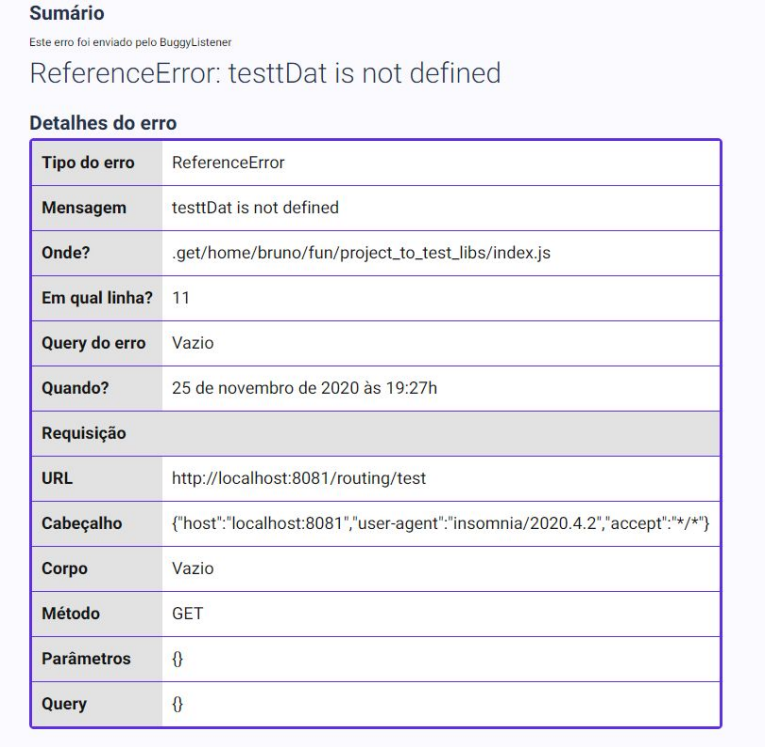
Fonte: Autor, 2020.

Esta funcionalidade é o centro de operações da plataforma. Nela pode-se ver os *bugs* por projeto, e verificar quais estão em aberto, quais estão sendo solucionados, em testes e que já foram fechados.

Um novo *bug* pode ser criado por qualquer desenvolvedor pertencente ao time, tem a habilidade de visualizar e alterar outros *bugs* mesmo que não não esteja designado à este.

3.1.4.2 Logger Listener

Figura 12: Exemplo de como um erro detectado pelo *Logger Listener* é mostrado para o usuário



Fonte: Autor, 2020.

É o diferencial do produto, que se consiste de uma biblioteca que roda dentro do projeto implementado pelo desenvolvedor, este é responsável por monitorar a aplicação em ambiente de produção, e quando um erro ocorre, ele detecta este erro e categoriza o mesmo de acordo com a gravidade do erro ocorrido, e então, automaticamente cria um *bug report* diretamente no projeto. É usada uma chave específica para protocolar o erro dentro do projeto no *Bug*gy.

3.1.5 Homologação do MVP

A base de validação foi baseada nos *feedbacks* e idéias que já são executadas e que já foram avaliadas em outras plataformas de gerência de projeto.

3.1.6 Comercialização do produto

A monetização do produto vai funcionar no modelo Freemium, o qual é um modelo de negócio dominante para serviços. O modelo se refere à estruturação de precificação de um produto onde o núcleo do serviço é gratuito, porém a receita é gerada através de vendas adicionais de produtos ou de funcionalidades *premium*.

Figura 13: Precificação da plataforma

Fonte: Autor, 2020.

Como analisado na Figura 12, o pacote gratuito vem com a possibilidade de ter apenas 1 (um) projeto, no máximo 5 (cinco) usuários adicionados à qualquer projeto em uma conta e 100MB de armazenamento. Enquanto no plano iniciante, custando R$100 (cem reais), será possível possuir 5 projetos criados na mesma conta, 1GB de armazenamento e adicionar até 20 pessoas nos projetos.

O pacote padrão com o custo de R$400 (quatrocentos reais) dará à empresa recursos ilimitados para usuários e até 100 (cem) projetos criados, além de 30GB em armazenamento. E por fim, o pacote *premium* com o custo de R$800 (oitocentos reais) dará à empresa recursos ilimitados para usuários e projetos criados, além de 100GB em armazenamento e possibilidade de criar campos personalizados para melhor gestão na plataforma.

Os preços cobrados se dão em torno de cálculos feitos com a quantia que será necessária para manter a plataforma. O servidor, hospedado pela *DigitalOcean*, inicialmente custará $120 (cento e vinte dólares), com um total de 16GB de memória RAM, 4 núcleos de CPU compartilhadas e até 5TB de transferência.

3.1.7 Produtos correlatos

A seguir serão apresentadas algumas das ferramentas com propostas parecidas com a do *Bug*gy, onde o módulo principal das mesmas se faz em torno do rastreamento de *bugs*.

3.1.7.1 Bugzilla

*Bug*zilla é um programa de rastreamento de *bugs* baseado na web de código aberto que, como o nome sugere, foi criado pela Fundação Mozilla. O programa foi desenvolvido pela Netscape em 1998, quando re-licenciou seu Navegador Netscape sob uma licença de código aberto como o pacote original do Mozilla. O software permite que os usuários enviem *tickets* e que os membros do projeto atribuam *bugs* a um nível de gravidade e atribuam *bugs* a desenvolvedores específicos (TECHOPEDIA., 2011)

Ainda a partir de dados da Techopedia (2011), *Bug*zilla foi originalmente escrito por Terry Wiseman em Tcl antes de ser re-implementado em Pearl. O sistema de rastreamento de *bugs* é baseado na web e roda em um sistema de gerenciamento de banco de dados e Perl 5. Foi desenvolvido primariamente para rastrear *bugs* em vários projetos da Mozilla, incluindo o navegador Firefox e o cliente de e-mail Thunderbird. É um exemplo de "*dogfooding*", onde uma empresa acaba usando os próprios produtos. Além do Mozilla, o *Bug*zilla também é usado para vários outros grandes projetos de código aberto, incluindo FreeBSD, WebKit, kernel do Linux e GNOME, entre outros.

3.1.7.2 Mantis Bug Tracker

A ferramenta Mantis *Bug* Tracker é uma opção web de rastreamento de defeitos baseada em linguagem PHP compatível com a maioria dos navegadores existentes. A sua instalação e configuração é o seu diferencial, quando se procura uma solução rápida e de baixo custo. É indicada tanto para desenvolvedores que precisam de um controle maior de suas atividades, gestores de projetos que necessitam gerenciar uma equipe ou para pessoas que desempenham diferentes papéis em um projeto de software (GUEDES, D. S.; TREPIM, D. M.; SOUZA, H.; PEREIRA, M. A., 2015).

Pode ser utilizada de duas formas diferentes, sendo sua versão desktop ou versão web indicadas segundo a necessidade do usuário final que serão explicadas nas próximas seções.

3.1.7.3 Jira

Jira é antes de tudo um software de gerenciamento de projetos, foi iniciado em 2004 como uma plataforma de rastreamento de problemas para desenvolvedores de software. Desde seu início, Jira cresceu para abranger todos os tipos de tipos de gerenciamento de projetos. (ATLASSIAN., 2004.)

A plataforma aproveita todos os tipos de habilidades de gerenciamento de projetos, incluindo desenvolvimento de software, gerenciamento de projetos Agile, rastreamento de *bugs*, gerenciamento de scrum, gerenciamento de conteúdo, marketing, gerenciamento de serviços profissionais e muito mais.O rastreamento de problemas é para o que Jira foi feito, e ainda se destaca como um rastreador de problemas. Usando o Jira, é possível identificar e criar novas tarefas de problema, anexar capturas de tela de problemas de software, priorizar problemas para conclusão e rastreá-los ao longo de seu fluxo de trabalho.

3.1.7.4 Diferenciais

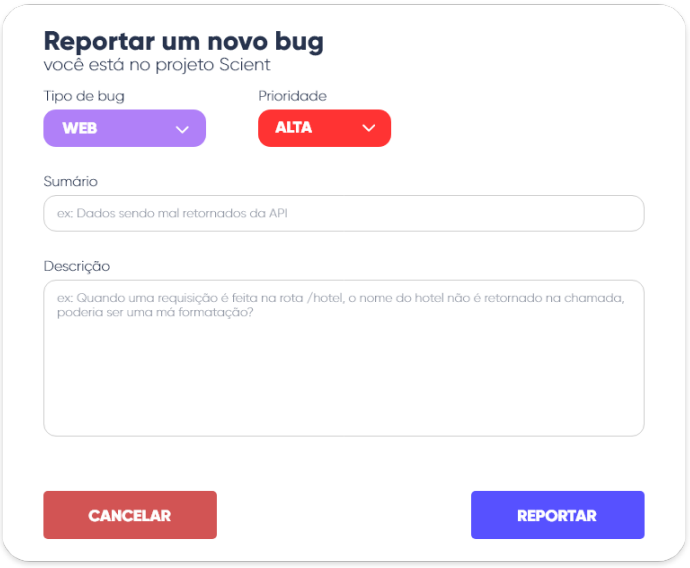
Todos os programas apresentados são aplicações conhecidas de rastreamento de falhas, e assim como o *Bug*gy, compartilham a mesma funcionalidade: um painel onde é possível pode verificar quais são os *bugs* existentes, ver quem foi designado para resolvê-los, tudo funcionando em formato Kanban - um sistema ágil e visual para gerência de tarefas (KNIBERG, H.; SKARIN, M., 2010). O que diferencia o presente produto dos demais, é a funcionalidade do Logger Listener, a capacidade de ter um módulo rodando no servidor do projeto de um usuário, e automaticamente registrar caso haja alguma falha não existe nos projetos citados.

3.2 Área de trabalho/abrangência

Por ser uma plataforma que trata sobre uma coisa em comum à todos os desenvolvedores, que são *bugs*, o projeto acaba abrangendo todos os times de desenvolvimento de software mundialmente, não tendo limitações geográficas para quem pode usá-lo.

3.3 Uso de arquivos de entrada e saída

Figura 14: Formulário para reportar um *bug*.



Fonte: Autor, 2020.

A principal funcionalidade do produto gira em torno dos *bugs*, é ela que move e dá sentido à aplicação. E nesta funcionalidade há duas maneiras de realizar uma entrada de dados, a primeira sendo pelo modo convencional, onde na plataforma, o *bug reporter* descreve manualmente qual o erro, qual o tipo do erro e qual a prioridade do erro.

Figura 15: Código responsável por ativar o Logger Listener

```
import * as buggyListener from 'buggy-listener-express';
import routes from './routes';

const app = express();

buggyListener.init();

app.use(routes);

app.use(buggyListener.requestError({ notInstanceOf: AppError }));
```

Fonte: Autor, 2020.

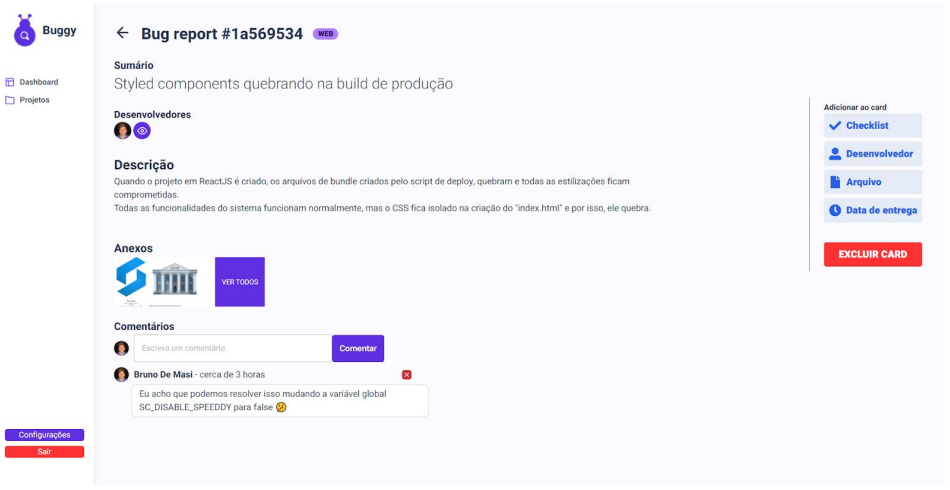
A outra maneira, como já citado no texto, se dá com o *Logger Listener*, responsável por monitorar erros que aconteçam em produção no projeto do cliente. Então, através de uma chave específica para cada projeto, o *Logger* faz uma requisição para a API do *Bug*gy, e adiciona esse erro que foi capturado como um não resolvido.

4 RESULTADOS

Nesta seção serão apresentados alguns resultados observados durante o todo o processo de desenvolvimento do projeto.

A prototipação da aplicação foi concluída com êxito nas primeiras duas *sprints* da idealização do produto. Foram definidas as telas, assim como o seu fluxo. A funcionalidade de *Bugs* foi a primeira a ser finalizada, e então a partir disso, o resto das telas foi desenvolvido.

Figura 16: Página de um *bug report*



Fonte: Autor, 2020.

Entretanto, no processo de desenvolvimento as telas sofreram reiterações para mudar o *design*. O layout e disposição dos componentes antigos continuaram distribuídos da mesma forma, porém foram aplicadas mais técnicas de UX (Experiência do Usuário) e UI (Interface do Usuário). Foi feita a adição de novos componentes como uma seção de “Adicionar ao Card” onde agora é possível adicionar módulos como checklist e anexar arquivos. Foi adicionada também uma seção de comentários que os desenvolvedores podem usar para interagir mais sobre o *bug* especificamente, além de compartilhar conhecimentos.

Já em relação ao desenvolvimento do módulo do *Logger Listener,* foi inicialmente implementado apenas com suporte para projetos desenvolvidos em Node usando Express (mais sobre no capítulo 2.3, onde são abordadas as Tecnologias usadas no projeto). Este módulo apresentou dificuldades ao abranger a captura de erros para diversas ferramentas de desenvolvimento, além da dificuldade na detecção, diferenciação e categorização dos erros.

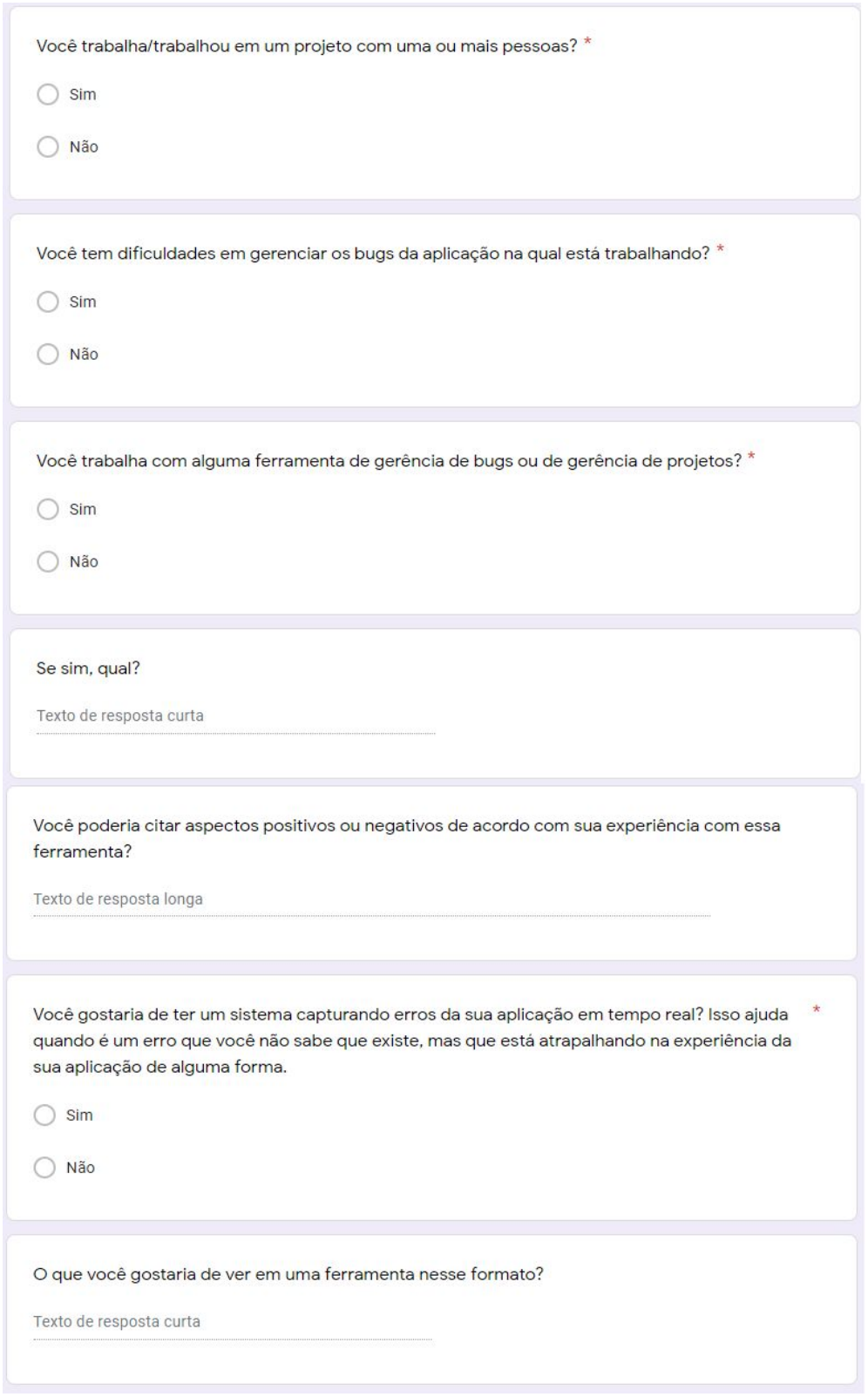
5 DISCUSSÃO

Nesta seção será apresentada uma discussão sobre os resultados obtidos, junto com dados de entrevistas que foram realizadas no formato de formulário digital.

De acordo com Krasner (2018), como exposto na introdução deste trabalho, uma aplicação relativamente pequena com 20.000 (vinte mil) linhas, terá em média 200 erros graves de codificação, sendo necessário, portanto, a atribuição de um método para dividir e priorizar a detecção e eventualmente correção desses *bugs*. Endossado por Alenezi e Banitaan (2013), até mesmo desenvolvedores mais competentes precisam de algum tipo de priorização desse *bug*, sendo possível, inclusive, a adoção de inteligência artificial conforme Naguib (2013).

Nesse sentido, priorizou-se a facilidade na interface e a necessidade de um tempo de resposta mais adequado com a identificação destes componentes. De acordo com Ahsan, Ferzundi e Wotawa (2008) recursos devem ser melhor explorados, diminuindo custos da empresa, de forma que pensou-se em uma pesquisa de mercado para comprovar ou refutar a adoção de ferramentas de identificação e correção de *bugs* no mercado local, dentro da área de atuação de desenvolvedores atualmente. Visto isso, se expôs um questionário de acordo com a estrutura da figura 16.

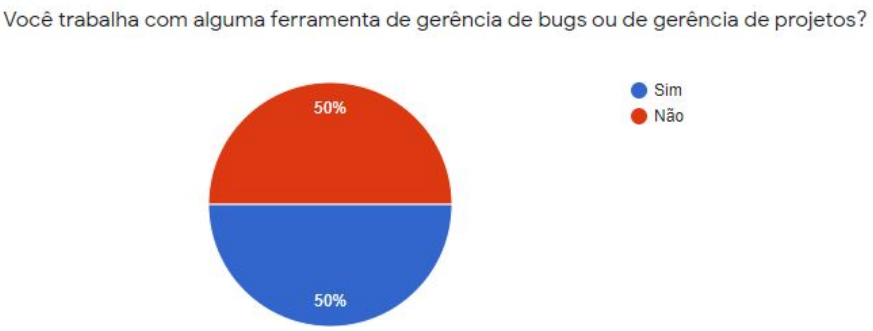
Figura 17: Formulário para pesquisa de mercado



Fonte: Autor, 2020.

Do questionário, foi pensado na formulação de questionamentos no sentido de identificar as principais ferramentas utilizadas e as dificuldades encontradas por desenvolvedores do mercado atual. Em análise da pesquisa, 50% dos entrevistados afirmaram não usar algum tipo de ferramenta para gerenciar projetos ou falhas, enquanto o restante afirmou que usa/já usou, citando ferramentas como: Trello®, Asana® e Azure DevOps®.

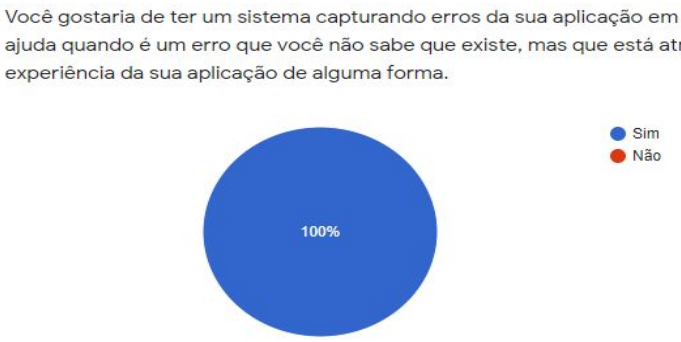
Figura 18: Respostas de uso de ferramenta de gerência



Fonte: Autor, 2020.

De acordo com as ferramentas apresentadas como alternativa, verifica-se que são ferramentas para gestão de projetos, não oferecendo um módulo de monitoramento e detecção de erros. Este módulo é interessante para os desenvolvedores participantes, visto que 100% deles (inclusive os que nunca trabalharam com qualquer ferramenta de gerência) responderam como afirmativo para o questionamento se o indivíduo, como desenvolvedor, gostaria de ter alguma ferramenta que monitore e capture erros em tempo real nas suas aplicações.

Figura 19: Respostas de uso de ferramenta de monitoramento



Fonte: Autor, 2020.

Pode-se considerar como motivos plausíveis para não adoção o valor normalmente atribuído a este tipo de solução, ou, ainda, a questões culturais, associadas a falta de costume de focar em melhoria de desempenho no processo de desenvolvimento de sistemas.

Também foram questionadas quais seriam as preferências do usuário em um módulo como o *Logger Listener,* e foram citadas opiniões como apenas mostrar o necessário, focando na simplicidade e efetividade do o quê, quando e como o erro aconteceu, o que já é um foco do módulo. Houveram também dicas para que a ferramenta fosse desenvolvida como uma entidade única e a parte, enquanto oferece integrações para plataformas de gerência de projetos ou falhas já existentes. Esta segunda parte no momento não é o foco projeto, porém é um ponto em consideração para trabalhos futuros.

Segundo XUAN, JIAN, REN e YAN (2017) é possível fazer com que os usuários possam colaborar para a plataforma gerando um relatório do *bug* do qual este possa ter encontrado. Já segundo Ahsan, Ferzund e Wotawa (2009), a resolução de um *bug* melhora a qualidade do software como um todo.

Entretanto, como Just, Premraj e Zimmermann (2008) citam, os relatórios que são necessários para gerar um *bug* report pode exigir muito do

usuário, visto que o mesmo não é familiarizado com as práticas de desenvolvimento, podendo gerar um *bug* report falho ou vago, acabando por atrapalhar na resolução precisa deste *bug*. O *Logger Listener* salta essa etapa, onde através da navegação do usuário, caso encontre um erro, o mesmo é detectado e enviado, sem precisar da escrita de um relatório.

6 CONCLUSÃO

Este Trabalho de Curso apresentou uma plataforma voltada para auxiliar desenvolvedores a monitorar, identificar e reportar de forma automática *bugs* de aplicações no ambiente de produção. Para o cumprimento dos requisitos necessários ao desenvolvimento do produto foi definido o escopo do sistema, bem como os requisitos de desenvolvimento da solução.

Outras etapas envolveram a modelagem do sistema utilizando princípios de *Scrum* e a criação de um método para detecção de *bugs* na aplicação, além de um ambiente para gerenciamento de *bugs*.

O projeto juntou dois conceitos já existentes em um só, no qual, pela pesquisa apresentada como validação da proposta, mostrou-se boa a adesão por parte dos desenvolvedores de uma ferramenta correlata.

O produto final mostrou-se funcional e alinhado às demandas de mercado, estando apto para a próxima etapa de disponibilização comercial. Foi, também, bem avaliado pela pesquisa de mercado exploratória para validação da proposta.

As principais dificuldades encontradas incluem a monitoração de erros em diversas tecnologias para que a captura destes erros seja mais abrangente. Outra dificuldade está associada a categorização e habilidade de conseguir rastrear qual foi a jornada do usuário até o momento em que o erro ocorreu.

Com intuito de capturar *feedback* dos usuários o produto está disponível para acesso, porém sem comercialização até o momento. Assim, pretende-se disponibilizar, como trabalho futuro, a validação da plataforma a partir de *beta testers* e após isso, finalmente, liberar a monetização da solução. Também, será pesquisado e implementado como trabalho futuro as integrações com ferramentas de controle de versão (GITHUB®), de comunicação (SLACK®, DISCORD®) e de gerência de projeto/falhas já existentes (TRELLO®, ASANA®).

7 REFERÊNCIAS

ANVIK, J.; HIEW, L.; MURPHY, G. C. Who should fix this *bug*? **Proceedings - International Conference on Software Engineering**, vol. 2006, p. 361–370, 2006. Disponível em <https://doi.org/10.1145/1134285.1134336>. Acesso em: 27/04/2020.

IBM. **What is *bug* tracking?** [2019?]. Disponivel em <https://www.ibm.com/topics/*bug*-tracking>. Acesso em: 27/04/2020

KRASNER, H. Quality Software A 2018 Report. **Consortium for IT Software Quality (CISQ)**, 2018.

TESTBYTES. **5 Major Benefits of Using a *Bug* Tracking System**. 2017. Disponível em: <https://www.testbytes.net/blog/*bug*-tracking-system/>. Acesso em 29/04/2020.

FIAZ, A. S. S.; DEVI, N.; AARTHI, S. *Bug* Tracking and Reporting System. no. 1, p. 42–45, 2013. Disponível em: <http://arxiv.org/abs/1309.1232>. Acesso em: 30/04/2020.

TILKOV, S.; VINOSKI, S. Node. js: Using JavaScript to build high-performance network programs. **IEEE Internet Computing**, vol. 14, no. 6, p. 80–83, 2010. Acesso em: 30/04/2020.

CHANIOTIS, I. K.; KYRIAKOU, K.-I. D.; TSELIKAS, N. D. Is Node. js a viable option for building modern web applications? A performance evaluation study. **Computing**, vol. 97, no. 10, p. 1023–1044, 2015. Acesso em: 30/04/2020.

O'DELL, J. **Why Walmart is using Node.js**. 2012. Disponível em <https://venturebeat.com/2012/01/24/why-walmart-is-using-node-js/**>**. Acesso em: 30/04/2020.

DATABASE STAR. **Why use Postgres**. 2019. Disponível em <https://www.databasestar.com/why-use-postgresql/>. Acesso em 30/04/2020.

DIEDRICH, C. **O que é docker?** 2015. Disponível em <https://www.mundodocker.com.br/o-que-e-docker/>. Acesso em 30/04/2020.

PANDIT, N. **What and Why React.js**. 2020. Disponível em <https://www.c-sharpcorner.com/article/what-and-why-reactjs/>. Acesso em: 30/04/2020.

JUST, Sascha; PREMRAJ, Rahul; ZIMMERMANN, Thomas. Towards the next generation of *bug* tracking systems. **Proceedings - 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008**, , p. 82–85, 2008. https://doi.org/10.1109/VLHCC.2008.4639063.

TATHAM, Simon. **How to Report *Bugs* Effectively.** 2008. Disponível em https://www.powerworld.com/files/weccconf_Aug2011/How%20to%20Report%20Software%20*Bugs*.pdf.

YIN, Zuoning; YUAN, Ding; ZHOU, Yuanyuan; PASUPATHY, Shankar; BAIRAVASUNDARAM, Lakshmi. How do fixes become *bugs*? , p. 26, 2011. https://doi.org/10.1145/2025113.2025121.

AHSAN, Syed Nadeem; FERZUND, Javed; WOTAWA, Franz. Automatic software *bug* triage system (bts) based on latent semantic indexing and support vector machine. 2009. **2009 Fourth International Conference on Software Engineering Advances** [...]. [*S. l.*]: IEEE, 2009. p. 216–221.

XUAN, Jifeng; JIANG, He; REN, Zhilei; YAN, Jun; LUO, Zhongxuan. Automatic *bug* triage using semi-supervised text classification. **arXiv preprint arXiv:1704.04769**, 2017.

GU, Zhongxian; BARR, Earl T; HAMILTON, David J; SU, Zhendong. Has the *bug* really been fixed? 1., 2010. **2010 ACM/IEEE 32nd International**

Conference on Software Engineering [...]. [*S. l.*]: IEEE, 2010. vol. 1, p. 55–64.

Mozilla Firefox. **Introdução Express/Node.** 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introdu%C3%A7%C3%A3o>.

Quero Bolsa. **Engenharia de Software no INATEL.** 2019. Disponível em: <https://querobolsa.com.br/inatel-instituto-nacional-de-telecomunicacoes/cursos/engenharia-de-software>.

AGGARWAL, Sanchit. Modern web-development using reactjs. **International Journal of Recent Research Aspects**, vol. 5, no. 1, p. 2349–7688, 2018. JALBERT, Nicholas; WEIMER, Westley. Automated duplicate detection for *bug* tracking systems. 2008. **2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)** [...]. [*S. l.*]: IEEE, 2008. p. 52–61.

BOEHM, Barry; BASILI, Victor R. Software defect reduction top 10 list. **Foundations of empirical software engineering: the legacy of Victor R. Basili**, vol. 426, no. 37, p. 426–431, 2005.

SUTHERLAND, Jeff. Business objects in corporate information systems. **ACM Computing Surveys (CSUR)**, vol. 27, no. 2, p. 274–276, 1995.

Associação Brasileira das Empresas de Softwares. **Brazilian Software Market 2019 Scenario and Trends**. 2019. Disponível em <http://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados2011/ABES-EstudoMercadoBrasileirodeSoftware-2019-Parcial-Ingles-Abr-2019.pdf>. Acesso em 22/09/2020.

LANDRE, E.; WESENBERG, H.; OLMHEIM, J. Agile enterprise software development using domain-driven design and test first. 2007. **Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion** [...]. [*S. l.*: *s. n.*], 2007. p. 983–993.

GEORGE, B.; WILLIAMS, L. A structured experiment of test-driven development. **Information and software Technology**, vol. 46, no. 5, p. 337–342, 2004.

MACHADO, LUIS. **Swift Test Driven Development (TDD)**. Medium**.** 2018. Disponível em <https://medium.com/@luisfmachado/swift-test-driven-development-tdd-810add46a1b9>. Acesso em 29 de set. 2020.

Guillaume Falourd. **Scrum: entenda os conceitos e como aplicar a metodologia.** Zup**.** 2019. Disponível em <https://www.zup.com.br/blog/scrum-conceitos-e-como-aplicar>. Acesso em 29 de set. 2020.

DOS SANTOS SOARES, M. Metodologias ágeis extreme programming e scrum para o desenvolvimento de software. **Revista Eletrônica de Sistemas de Informação**, vol. 3, no. 1, 2004.

KRASNER, H. The cost of poor quality software in the us: A 2018 report. **Consortium for IT Software Quality, Tech. Rep**, vol. 10, 2018.

PARASOFT. What Do Defects Really Cost? 2015. Disponível em <https://cdn2.hubspot.net/hubfs/69806/Reassessing_the_Cost_of_Software_Quality.pdf?t=1431566550782>. Acesso em 30 de set. 2020.

JONES, C. **Applied software measurement: global analysis of productivity and quality.** [S. l.]: McGraw-Hill Education Group, 2008.

NIELSEN, J. **Ten usability heuristics.** 2005.

ALENEZI, M.; BANITAAN, S. ***Bug* reports prioritization: Which features and classifier to use?** 2., 2013. 2013 12th International Conference on Machine Learning and Applications [...]. [S. l.]: IEEE, 2013. vol. 2, p. 112–116.

NAGUIB, H.; NARAYAN, N.; BRÜGGE, B.; HELAL, D. ***Bug* report assignee recommendation using activity profiles.** 2013. 2013 10th Working Conference on Mining Software Repositories (MSR) [...]. [S. l.]: IEEE, 2013. p. 22–30.

EVANS, E.; DESIGN, D.-D. **Tacking Complexity In the Heart of Software.** 2003.

KNIBERG, H.; SKARIN, M. **Kanban and Scrum-making the most of both**. [S. l.]: Lulu. com, 2010.

GUEDES, D. S.; TREPIM, D. M.; SOUZA, H.; PEREIRA, M. A. **Plugin Mylyn: conheça o Mantis**. 2015. Disponível em <https://www.devmedia.com.br/plugin-mylyn-conheca-o-mantis/37579>.

ATLASSIAN. **Para que serve o Jira?**. 2004. Disponível em<https://www.atlassian.com/br/software/jira/guides/use-cases/what-is-jira-used-for#Jira-for-requirements-&-test-case-management>

TECHOPEDIA. ***Bug*zilla**. 2011. Disponível em <https://www.techopedia.com/definition/18111/*bug*zilla>.

MARKET DATA FORECAST. Impacts of COVID-19 on the Information Technology (IT) industry. 2020. Disponível em: https://www.marketdataforecast.com/blog/impacts-of-covid19-on-information-technology-industry.