UNIVERSIDADE DO SUL DE SANTA CATARINA

FELIPE AMILIBIA BRAGA

QUALIDADE DE SOFTWARE:
PROPOSTA DE AUTOMAÇÃO DE TESTES E DE UM PROCESSO ÁGIL PARA UMA EMPRESA DE SOFTWARE

Florianópolis

2018

UNIVERSIDADE DO SUL DE SANTA CATARINA

FELIPE AMILIBIA BRAGA

QUALIDADE DE SOFTWARE:
PROPOSTA DE AUTOMAÇÃO DE TESTES E DE UM PROCESSO ÁGIL PARA UMA EMPRESA DE SOFTWARE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação.

Prof. Maria Inés Castiñeira (Orientadora)

Florianópolis

2019

FELIPE AMILIBIA BRAGA

QUALIDADE DE SOFTWARE: PROPOSTA DE AUTOMAÇÃO DE TESTES E DE UM PROCESSO ÁGIL PARA UMA EMPRESA DE SOFTWARE

Este trabalho de conclusão de curso foi julgado adequado à obtenção do título de Bacharel em Sistemas de Informação e aprovado em sua forma final pelo Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina.

Florianópolis, 20 de novembro de 2019

Maria Inés Castiñeira, Dra. (Orientadora)

Universidade do Sul de Santa Catarina

Aran Morales, Dr.

Universidade do Sul de Santa Catarina

Edson O. Lessa, Esp.

Universidade do Sul de Santa Catarina

A todos os professores do curso, que foram tão importantes na minha vida acadêmica e no desenvolvimento desta monografia.

AGRADECIMENTOS

A professora Maria Inés, minha orientadora, por ter acreditado na possibilidade da realização deste trabalho, pela sua paciência e seu encorajamento, pela disponibilidade e sugestões que foram preciosas para a concretização desta monografia.

Aos meus pais, que me incentivaram e acreditaram no potencial da realização deste trabalho, com eles compartilho esse momento que é o mais importante de minha realização acadêmica.

“A persistência é o caminho do êxito.” (Charles Chaplin).

RESUMO

O presente trabalho tem como objetivo apresentar uma proposta de automação de teste e adoção de uma metodologia ágil de desenvolvimento de software em um projeto de uma empresa estudo de caso. A proposta tem por intuito apoiar o processo de desenvolvimento, com vistas a torná-lo mais ágil, acrescentando qualidade através da solução de problemas relacionados a entrega do projeto e crescente demanda de manutenção de erros nas aplicações desenvolvidas. Inicialmente foi realizada uma pesquisa bibliográfica, através de estudos de cursos, livros e artigos científicos sobre engenharia de software, qualidade de software, testes de software, processos e estratégias de teste, e metodologias ágeis para fundamentar a aplicação da automação de testes e da metodologia ágil de desenvolvimento de maneira correta e eficaz. O trabalho se baseia em uma proposta de mudança para a empresa estudada. Primeiro foi realizada a descrição da empresa, quantidade de colaboradores, seus respectivos núcleos de projeto. A seguir, foi apresentado seu processo desenvolvimento, que é considerado tradicional, na sequência foi detalhado a proposta do novo processo, sendo que este é baseado em metodologias ágeis e automação de testes. A proposta de processo é focada no planejamento e entrega do produto com qualidade e confiança, e a automação de testes objetiva diminuir o retrabalho e aumentar de maneira considerável a comunicação entre a equipe através de uma linguagem de fácil entendimento para analistas, clientes e desenvolvedores. Foram apresentadas as desvantagens do processo de desenvolvimento antigo e as vantagens e melhorias que a proposta pode proporcionar. Por exemplo, sabe-se que o desenvolvimento orientado por comportamento, conhecido pela sigla de BDD, pode representar melhorias na questão de comunicação e confiança da equipe e no tempo de desenvolvimento e testes quando utilizada automação de testes.

Palavras-chave: Automação de Testes. BDD. *Behaviour Driven Development.* Metodologia de desenvolvimento de software ágil.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Andrade (2015, p. 10) afirma que a exigência da qualidade de *software* está se tornando o centro das atenções para o desenvolvimento de *software*, não é mais uma vantagem no mercado, mas sim uma condição necessária e indispensável para que a empresa consiga competir com sucesso em um mercado que está em constante desenvolvimento.

Para assegurar a qualidade de software é importante assegurar a qualidade do processo de software. Mas o que é processo de software?

Conforme citado por Pressman (2011, p. 40), um processo não é uma determinação rígida de como desenvolver um *software*. Ao contrário, é um comportamento flexível que possibilita aos integrantes da equipe realizar o trabalho de distinguir e adotar o conjunto adequado de ações e tarefas.

Para que o desenvolvimento de um *software* seja coerente com os requisitos adotados e possua qualidade, poderão ser adotados modelos de processos, que segundo Pressman (2011, p. 41) são um conjunto de métodos organizados/planejados para um fim específico.

Bartié (2002, p. 18) afirma que uma abordagem para se medir a qualidade do produto é a realização de testes, como por exemplo teste caixa branca, caixa preta, funcional, de performance, entre muitos outros. A realização dos mesmos é de suma importância para avaliar a qualidade do produto.

Em alguns casos, quando os testes são repetitivos, pode ser apropriado aplicar automação de testes.

Bernardo e Kon (2008) afirmam que a automação de testes é algo amplamente utilizado pelo mercado, principalmente em metodologias ágeis, com o foco de agilizar o processo de testes garantindo que tarefas repetitivas e que levariam uma estimativa maior sejam realizadas em um curto período e permitindo ao *tester* focar em outras tarefas e que não poderiam ser automatizadas.

Conforme Bartié (2002, p. 188) “as ferramentas de automação dos testes possibilitam o desenvolvimento de *scripts* automatizados, de forma a viabilizar um processo de teste com as atividades de entrada e conferência de valores totalmente automatizados”.

1.1 PROBLEMÁTICA

Este trabalho aborda um estudo de caso na área de automação de testes de software no processo de desenvolvimento ágil em uma empresa da área da saúde na grande Florianópolis.

Atualmente a empresa estudada investe boa parte de seus esforços e investimentos na resolução de *bugs* encontrados em seus projetos. Esse cenário relacionado aos testes acontece enquanto a equipe de desenvolvimento está se adaptando a um modelo de processo ágil.

Estes mesmos bugs, relatados pelos clientes e funcionários da empresa, podem ser originados devido ao processo de desenvolvimento e as responsabilidades dos papeis de cada membro da equipe não estarem bem definidas.

A ausência de ferramentas para automação de testes dificulta a liberação dos produtos feitos pela empresa, fazendo com que o processo da equipe de Qualidade seja demorado e trabalhoso. Esse cenário relacionado aos testes acontece enquanto a equipe de desenvolvimento está migrando para uma metodologia de desenvolvimento ágil. Assim, a questão de pesquisa que norteia este trabalho é: como a automação de testes pode ser adotada na migração para um processo de desenvolvimento de software ágil?

1.2 OBJETIVOS

Serão apresentados a seguir, os objetivos do trabalho.

1.2.1 OBJETIVO GERAL

Apresentar recomendações para adotar automação de testes junto com um modelo de desenvolvimento ágil em um projeto da empresa estudada.

1.2.2 OBJETIVOS ESPECÍFICOS

- Pesquisar processos de desenvolvimento de *software* que podem ser utilizados em metodologias ágeis
- Estudar tecnologias e ferramentas de automação de testes.
- Listar principais problemas no processo de software do projeto da empresa estudada.
- Planejar e propor a automação de testes.
- Gerar as recomendações para o modelo de processo junto com a automação de testes.

1.3 JUSTIFICATIVA

Como dito anteriormente, a qualidade de *software* é um fator imprescindível para o sucesso da empresa. Além de entregar o produto na data requerida e com os requisitos sendo cumpridos, é necessário ser levado em conta o cliente, usabilidade, confiabilidade, eficiência, entre outros fatores que garantam que o produto seja feito com qualidade.

> A qualidade tem sido um fator de diferenciação no mercado atual. A exigência por qualidade tem aumentado em todas as áreas e afetado também a indústria de software. Com os computadores, cada vez mais, fazendo parte da vida das pessoas, a produção de softwares vem aumentado e tornando os clientes mais exigentes. A grande exigência dos clientes por melhores softwares tem obrigado os desenvolvedores a aperfeiçoarem o seu produto final para continuarem competindo no mercado. (ANDRADE, 2015, p. 7)

Segundo Bartié (2002), atualmente toda a empresa de *software* que pretende manter-se no mercado de desenvolvimento possui uma metodologia para realizar testes em seu produto para garantir que o mesmo além de preencher os requisitos e o cronograma estipulados, esteja com alta qualidade.

> Entender que o objetivo dos testes é “provar que algo não funciona” é um avanço significativo na compreensão de um processo de qualidade de *software*. Não adianta termos documentações incompletas, imprecisas e ambíguas. É necessário buscar um maior nível de qualidade em todos os produtos (sejam documentos ou *software*s) produzidos em todas as fases do ciclo de desenvolvimento. Esses documentos auxiliarão as equipes a tomarem decisões mais corretas sobre o projeto de *software*

> que refletirá em um produto com maior conformidade com as necessidades dos clientes e usuários. (BARTIÉ, 2018, p. 21)

Segundo Andrade (2015, p. 7), ao realizar o aprimoramento da qualidade do processo de software, consequentemente aumentam-se as chances de adquirir um produto final mais adequado e que esteja a altura das expectativas do cliente. Para ser alcançado esse aprimoramento existem inúmeras abordagens que descrevem como a empresa pode avaliar o estado atual e com o resultado dessa avaliação procurar melhorar seu processo de maneira ideal e objetiva.

1.4 ESTRUTURA DE MONOGRAFIA

Este trabalho está dividido em 5 capítulos. Segue abaixo sua estrutura:

- Capítulo 1 – Introdução, problemática e os objetivos (geral e específico).
- Capítulo 2 – Apresenta conceitos e temas relacionados a Engenharia de *software*, Qualidade de *software*, testes de *software* e seus tipos, automação de testes e BDD.
- Capítulo 3 – Metodologia
- Capítulo 4 – Desenvolvimento da proposta e descrição da empresa
- Capítulo 5 – Conclusões e trabalhos futuros

2 REVISÃO DA LITERATURA

Neste capítulo é abordada a revisão da literatura. São estudados temas de engenharia de *software*, qualidade de *software*, testes, tipos de testes, entre outros.

2.1 ENGENHARIA DE SOFTWARE

Engenharia de *software* segundo Sommerville (2011) é a área que abrange todo o processo de criação e de desenvolvimento do *software*, tais como os requisitos do sistema levantados pelo responsável do projeto até seu lançamento e entrega, e futuramente sua manutenção.

"Engenharia tem a ver com obter resultados de qualidade requeridos dentro do cronograma e do orçamento. Isso frequentemente envolve ter compromissos — engenheiros não podem ser perfeccionistas". (SOMMVERVILLE, 2011, p. 5)

Em suma, os engenheiros de *software* são metódicos e regrados em relação aos seus processos de desenvolvimento, seguindo de certa forma um padrão com foco em eficiência e na produção de alta qualidade.

Conforme Sommerville (2011, pag. 5), engenharia está relacionada a escolher um processo mais adequado para desenvolvimento do produto solicitado, então uma abordagem mais inovadora e menos abordada pode ser mais eficaz em algumas circunstâncias.

> No contexto da engenharia de *software*, um processo não é uma prescrição rígida de como desenvolver um *software*. Ao contrário, é uma abordagem adaptável que possibilita às pessoas (a equipe de *software*) realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas. A intenção é a de sempre entregar o *software* dentro do prazo e com qualidade suficiente para satisfazer aqueles que patrocinaram sua criação e àqueles que irão utiliza-lo. (PRESSMAN, 2011, p. 40)

Portanto, mesmo que o processo de desenvolvimento preferido pelo desenvolvedor seja considerado eficaz na maioria dos casos, em certos aspectos e situações não previstas no planejamento, adotar um método de desenvolvimento novo e/ou alternativo pode ser mais apropriado.

Segundo Sommerville (2011, p. 39), na década de 1980 a metodologia adotada para o desenvolvimento de um *software* considerado de alta qualidade era através de uma análise e elaboração minuciosa de todo o projeto, formalização da segurança e padronização do processo de desenvolvimento.

> Na economia moderna é frequentemente difícil ou impossível prever como um sistema computacional (por exemplo, uma aplicação baseada na Web) irá evoluir com o tempo. As condições de mercado mudam rapidamente, as necessidades dos usuários finais se alteram e novas ameaças competitivas emergem sem aviso. (PRESSMAN, 2011, p. 82)

Para Sommerville (2011, p. 39), devido a essa metodologia rígida e com processo de desenvolvimento considerado lento, um grupo de desenvolvedores propuseram novas metodologias com sua essência focada no desenvolvimento de seus produtos e os nomearam de ‘métodos ágeis’.

> Eles são mais adequados ao desenvolvimento de aplicativos nos quais os requisitos de sistema mudam rapidamente durante o processo de desenvolvimento. Destinam-se a entregar o *software* rapidamente aos clientes, em funcionamento, e estes podem, em seguida, propor alterações e novos requisitos a serem incluídos nas iterações posteriores do sistema. (SOMMERVILLE, 2011, p. 40)

Segundo Sommerville(2011, p. 40), a ideologia por trás destes métodos ágeis foi repercutida no manifesto ágil, que afirma:

- Indivíduos e interações estão acima de processos e ferramentas
- *Software* em funcionamento estão acima de documentação abrangente
- Colaboração do cliente acima de negociação de contrato
- Respostas a mudanças acima de seguir um plano

Podemos listar inúmeras metodologias ágeis que foram criadas e são utilizadas atualmente, tais como *Scrum*, *Extreme Programming (XP)* e *Kanban*. Neste trabalho abordaremos detalhadamente as metodologias *Scrum* e *Kanban*, pois são os modelos de interesse de adoção pela empresa estudada.

2.1.1 SCRUM

Conforme Pressman (2011, p. 95), *Scrum* é uma metodologia de desenvolvimento ágil que tem sido usado desde a década de 90 para examinar e solucionar problemas considerados complexos de uma maneira simples e eficaz, usufruindo de inúmeras técnicas e processos para serem utilizados na resolução do mesmo problema.

> Os princípios do *Scrum* são consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades estruturais: requisitos, análise, projeto, evolução e entrega. Em cada atividade metodológica, ocorrem tarefas a realizar dentro de um padrão de processo chamado *Sprint*. (PRESSMAN, 2011, p. 95)

Sommerville (2011, p. 50) declara que o *Scrum* pode ser dividido em 3 etapas, e as descreve da seguinte maneira:

- A primeira é caracterizada como a fase de planejamento geral, em que se estabelecem os objetivos gerais do projeto e da arquitetura do software.
- Em seguida, ocorre uma série de ciclos de *sprint*, sendo que cada ciclo desenvolve um aprimoramento do sistema.
- A última fase do projeto é caracterizada pelo encerramento do projeto, a documentação requerida é completada, como quadros de ajuda do sistema e manuais do usuário, e é realizada uma avaliação das lições aprendidas com o projeto para que problemas que tiverem ocorrido durante o projeto, não ocorram novamente.

Sommerville (2011, p. 50) complementa afirmando que o principal fator para o planejamento bem-sucedido do projeto é o desenvolvimento do *backlog* do produto, que compõe todas as tarefas e funcionalidades que deverão ser realizadas no decorrer projeto. O cliente está intimamente envolvido no processo de criação do *backlog*, além disso podem ser introduzidos novos requisitos ou tarefas pelo mesmo no início da *sprint.*

Schwaber (2013, p. 6) afirma que o pilar principal que sustenta o *Scrum* é a *sprint*, ela aborda o conceito do *time-boxed*, que é o intervalo de tempo para a realização de uma determinada atividade, de no máximo 30 dias, durante este tempo será produzido um "*done*" que representa uma versão utilizável do produto.

Segundo Sutherland (2013, p. 3), o *Scrum* é composto pelo Time Scrum que se resume ao *Product Owner*, o Time de Desenvolvimento e ao *Scrum Master*, sendo que cada integrante do Time é ligado a um determinado papel, evento e/ou regra.

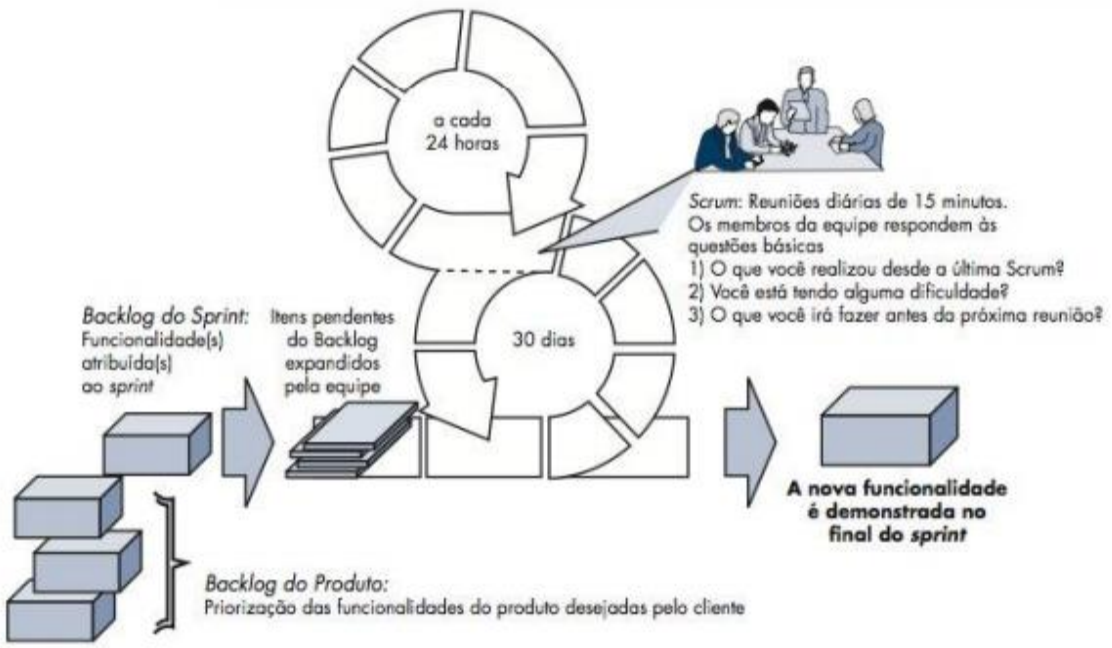
“Times Scrum são auto-organizáveis e multifuncionais. Times auto-organizáveis escolhem qual a melhor forma para completarem seu trabalho, em vez de serem dirigidos por outros de fora do Time.” (SCHWABER; SUTHERLAND, 2013, p. 5)

Conforme Schwaber (2013, p. 5), o *Product Owner* tem a responsabilidade de alavancar o valor do produto e do comprometimento do Time de Desenvolvimento. A maneira que isso será feito irá depender da empresa, do Time Scrum e dos colaboradores envolvidos no projeto.

O Time de Desenvolvimento é composto por colaboradores que exercem a função de entregar uma versão utilizável do produto solicitado a cada final de *sprint*.

E o *Scrum Master* possui o dever de fazer com que toda a equipe compreenda e que use corretamente o *scrum*, é visto como um facilitador e auxiliador para que todos consigam realizar o trabalho para que no final da *sprint* todas as demandas sejam adequadas ao que foi solicitado.

Figura 1 - Fluxo geral do *scrum*



Fonte: PRESSMAN (2011, p. 96)

2.1.2 KANBAN

A metodologia de desenvolvimento de *software* kanban de acordo com Mariotti (2012, p. 6) se baseia em nunca sobrecarregar o colaborador da empresa moderando o número de atividades que o mesmo pode iniciar, com o princípio de que uma nova tarefa poderá ser iniciada somente após a tarefa ou item em andamento ser concluído.

A palavra “kanban” é de origem japonesa, e sua tradução seria “sinal” ou “cartão”, a metodologia foi denominada dessa maneira, pois possui como um dos principais objetivos sinalizar de maneira clara e de fácil entendimento o andamento do projeto para todos os membros da equipe envolvida, assim mostrando através de cartões o status das tarefas, e sinalizando quando uma nova tarefa poderá ser inicializada.

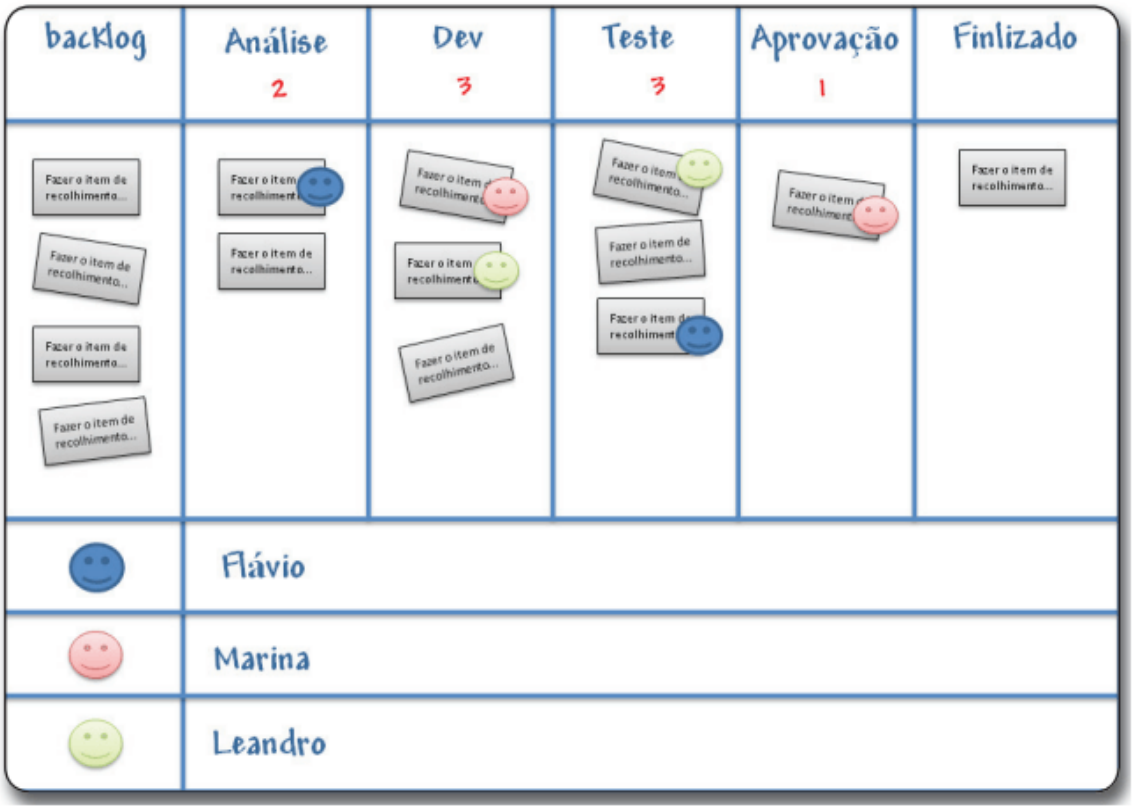
> Uma boa metáfora que descreve essa regra é imaginarmos uma rodovia que suporta até 100 veículos para manter o fluxo de trafego com um bom desempenho, porém em todos os feriados essa rodovia recebe em torno de 200 veículos. Essa demanda não suportada pela rodovia gera um congestionamento afetando consideravelmente o desempenho do trafego. Logo, não adianta empurrar um número de atividades não suportada pela equipe, isso irá causar um “congestionamento” e afetar o desempenho de produção. (MARIOTTI, 2012, p. 7)

Segundo Mariotti (2012, p. 7) o kanban é resumido em três princípios:

- Visualização de processos;
- Limitação de trabalho em processo, mais conhecido como WIP (Work *in progress*)
- Gerenciamento do tempo que a atividade leva para ser finalizada, conhecido como *lead-time*

A seguir é apresentada a figura de utilização de kanban de um quadro de uma empresa:

Figura 2 - Ilustração de sinalizador visual Kanban



Fonte: MARIOTTI (2012, p. 9)

Mariotti (2012, p. 8) afirma que muitas empresas começaram a utilizar o kanban por ser uma metodologia que proporciona um equilíbrio organizacional para a equipe, graças a limitação de tarefas e assim não sobrecarregando seus membros. Outra vantagem ao utiliza-la é que é uma metodologia de fácil implementação e entendimento, pois utiliza o sistema de cartões podendo ser usado em lousas, quadros, paredes, tabuleiros, entre outros, dessa forma apresentando de maneira clara o progresso do projeto e suas prioridades.

As vantagens alcançadas por empresas que adotaram o Kanban são listadas por Mariotti:(2012, p. 9)

- Falhas tornam-se claramente visíveis em tempo real;
- O benefício de encontrar os “gargalos” faz com que as pessoas passem a colaborar ainda mais para a cadeia de valor em vez de apenas fazerem a sua parte.

- Fornece uma evolução gradual do processo cascata para o modelo de desenvolvimento ágil de *software*.

2.2 QUALIDADE DE SOFTWARE

Há inúmeras interpretações para o conceito de qualidade propostas na literatura, sob pontos de vistas diferentes. Segundo Martinelli (2009, p. 15) não seria correto termos uma conclusão final sobre qual o conceito do termo qualidade, pois se levarmos em conta diversos profissionais de diversas áreas, cada um concederia uma resposta diferente.

Na área de *software* autores consagrados ponderam sobre a qualidade do produto de *software* e a qualidade do processo.

> Um pressuposto do gerenciamento de qualidade de *software* é que a qualidade do *software* é diretamente relacionada à qualidade do processo de desenvolvimento de *software*. Isso vem novamente da fabricação de sistemas, em que a qualidade de produto é intimamente relacionada ao processo de produção. (SOMMERVILLE, 2011, p. 457)

Para Pressman (2011, p. 359) na área de desenvolvimento de *software*, a qualidade de *software* é definida por uma gerência de qualidade efetiva aplicada de uma maneira que ofereça um produto que agregue um valor apreciável para aqueles que o desenvolveram e para aqueles que o solicitaram.

Conforme Sommerville (2011, p. 455) a preocupação com o gerenciamento de qualidade do produto fornece uma verificação precisa e independente do processo de desenvolvimento de *software*. O gerenciamento de qualidade é caracterizado por realizar a verificação de quais funcionalidades do projeto já estão prontas, e com isso verificar se as mesmas estão de acordo com os requisitos e padrões solicitados pelo cliente.

Assim, conforme Sommerville (2011, p. 455) a equipe de QA (*Quality Assurance*) deve agir de maneira separada da equipe de desenvolvimento para que suas análises e visões sobre o produto sejam imparciais em relação ao desenvolvimento do *software*.

Nos modelos ágeis, conforme apresentado na metodologia *Scrum* existe uma maior interação entre os desenvolvedores e a equipe de QA. Por exemplo nesse modelo todos eles integram o time *scrum*.

2.3 TESTE DE SOFTWARE

Para Pressman (2011, p. 402.) Testes de *software* são um grupo de atividades construídas e aplicadas metodicamente antes mesmo do desenvolvimento do *software* estar concluído, com o propósito de verificar se o produto realiza as ações requisitadas pelo cliente, desde sua interface e layout até mesmo se o código fonte foi implementado corretamente.

> O teste é destinado a mostrar que um programa faz o que é proposto a fazer e para descobrir os defeitos do programa antes do uso. Quando se testa o *software*, o programa é executado usando dados fictícios. Os resultados do teste são verificados à procura de erros, anomalias ou informações sobre os atributos não funcionais do programa. (SOMMERVILLE, 2011, p. 144)

2.3.1 Processo de teste de software

De acordo com Sommerville (2011, p. 144) o processo de testes possui dois objetivos finais, o primeiro é validar e comprovar para o cliente e para o desenvolvedor que o *software* cumpre todos os requisitos e regras de negócios solicitadas, isso por meio de testes singulares para cada requisito solicitado e testes que validem as características do sistema como códigos, layouts e interfaces. O segundo objetivo é encontrar situações e cenários em que o *software* não se comporte de maneira adequada, ou seja, não conforme sua documentação. Essas situações são descritas como os defeitos do *software*, e esses testes dos defeitos tem como foco expor atitudes indesejadas da aplicação, como erros na tela, dados incorretos e interfaces fora do padrão.

Testes de *software* podem ter diferentes conceitos pelos membros da equipe, conforme demonstra da figura:

Figura 3 – Simplificação das definições de teste

Fonte: BARTIÉ (2002, p. 20).

Conforme Bastos (2007, p. 20) atualmente o teste não pode mais ser um acréscimo no processo de desenvolvimento da aplicação, é indispensável que faça parte do processo de desenvolvimento, integrado com esse processo. As atividades do Analista de QA iniciam quando a funcionalidade começa a ser desenvolvida, com isso é possível obter melhores resultados na garantia de qualidade do software.

Bartié (2002, p. 20) afirma que o conceito geral de testes de software é visto como uma maneira de comprovar que toda a aplicação está funcionando corretamente conforme as solicitações do cliente, de que não há problemas e que o cenário feliz do software funciona. Na verdade a complexidade de demonstrar que algo na aplicação não funciona, realizar um cenário não previsto ou verificar um comportamento inesperado, é muito maior do que simplesmente mostrar o que funciona no sistema.

Segundo Bastos (2007, p.20) com a execução do processo de testes ocorrendo de maneira paralela ao de desenvolvimento, os custos necessários para a correção de defeitos que seriam detectados em produção futuramente caem drasticamente. O custo do desenvolvimento do software, que é baseado em seu desenvolvimento e manutenção, será relativamente menor quando a aplicação é testada de maneira correta.

2.3.2 Estratégias de teste de software

Pressman (2011, p. 401) afirma que a estratégia de teste de *software* descreve um plano que estabelece todas as etapas que devem ser executadas nos processos de teste, indica

em que momento os testes serão planejados, quando os mesmos serão executados e o quanto de trabalho e recursos da equipe que serão exigidos. Pressman (2011, p. 401) conclui que as estratégias de teste de *software* devem ser maleáveis, a ponto de permitirem que novas técnicas e abordagens inovadoras sejam adotadas durante o decorrer dos testes, porém ao mesmo tempo afirma que devem ser consistentes o bastante a ponto de promover seu planejamento lógico e a orientação mediante ao progresso do projeto.

Segundo Shooman (1983, apud PRESSMAN, 2011, p. 401):

> De muitas formas, o teste é um processo individualista e o número de tipos diferentes de testes varia tanto quanto as diferentes abordagens de desenvolvimento. Por muitos anos, nossa única defesa contra os erros de programação era um projeto cuidadoso e a inteligência do programador. Estamos agora em uma era na qual as modernas técnicas de projeto [e revisões técnicas] estão nos ajudando a reduzir a quantidade de erros iniciais inerentes ao código.

Segundo Luís (2018, p. 4) os testes realizados pela equipe de QA podem ser classificados em inúmeros tipos, entretanto existem os que são considerados essenciais em qualquer a estratégia de software:

- Teste de sistema: É um tipo de teste desenhado para testar a aplicação em sua plenitude, é executado depois da aplicação estar completamente integrada e concluída para serem validados os requisitos funcionais.
- Teste de validação: É classificado como um grupo de testes que engloba os testes unitários, testes de integração e de aceitação.
- Teste de integração: É o teste que visa em demonstrar a resistência do *software* a falhas, onde são unidos módulos individuais, que estejam funcionando corretamente, para serem testados em grupo e verificar possíveis defeitos.
- Teste unitário ou de unidade: É um tipo de teste que tem o foco de testar uma única funcionalidade, aplicação ou característica do produto.

A seguir é apresentada a figura que ilustra a visão ampla da estratégia de teste de *software*:

Figura 4 - Estratégia de teste

Fonte: PRESSMAN (2011, p. 404)

Nos próximos tópicos são abordados os conceitos sobre os tipos de testes de *software*.

2.3.2.1 Teste de caixa branca (*White Box*)

Pressman (2011, p. 431) afirma que os testes de caixa branca, também conhecidos como testes estruturais, é a estratégia de teste que tem o foco em testar o código do produto, gerando casos de testes que:

- Garantam que todos os caminhos independentes de um módulo foram exercitados pelo menos uma vez

- Exercitam todas as decisões lógicas nos seus estados verdadeiro e falso
- Executam todos os ciclos em seus limites e dentro de suas fronteiras operacionais
- Exercitam estruturas de dados internas para assegurar sua validade

Muitas vezes estes testes são realizados pelo próprio desenvolvedor.

2.3.2.2 Teste de caixa preta (*Black Box*)

Pressman (2011, p. 439) declara que o teste de caixa preta, também conhecido como teste comportamental, é a estratégia de software que tem o foco nos requisitos funcionais da aplicação, testando os comportamentos da aplicação em diferentes cenários de teste para serem verificados erros nas seguintes categorias:

- Funções incorretas ou faltando
- Erros de interface
- Erros em estrutura de dados ou acesso a base de dados externas
- Erros de comportamento ou desempenho
- Erros de inicialização ou término

2.3.3 Tipos de teste

Conforme Luís (2018, p. 4) durante o ciclo de processo de desenvolvimento de *software* até o mesmo estar concluído, são realizados inúmeros testes pela equipe de desenvolvimento e pela equipe de qualidade para garantir que o produto esteja de acordo com os requisitos solicitados pelo cliente. Neste trabalho são listados os seguintes tipos de teste:

- Teste funcional: Este teste é caracterizado por possuir o objetivo de validar se as funções disponíveis pelo *software* estão de acordo com o especificado pelo cliente na documentação do produto, como por exemplo requisitos de negócios e requisitos técnicos.
- Teste de carga: É o teste que tem como objetivo simular a carga em condições de normais de uso.
- Teste de instalação: Tipo de teste que verifica se o *software* é instalado corretamente em diferentes hardwares sob condições adversas como pouca memória, internet ruim e interrupções durante a instalação.
- Teste de segurança: São testes focados em verificar se o sistema funciona de maneira segura e se os dados estão protegidos.
- Teste de volume: É o teste que visa simular o comportamento do sistema com em períodos de maior tráfico de dados esperado, e com isso monitorar os impactos na rede.
- Teste de *stress*: Teste que tem como objetivo verificar o comportamento do *software* quando se atinge o máximo (ou além disso) do tráfico de dados suportados.
- Teste de regressão: Um dos testes considerados mais importantes, é o reteste de funcionalidades anteriormente desenvolvidas para verificar se alguma modificação ou nova funcionalidade impactou nas mesmas.
- Teste de manutenção: É o teste que verifica se a mudança de ambiente, exemplo ambiente de pré-produção para ambiente de produção, gerou impacto no funcionamento do sistema.
- Teste de usabilidade: É o teste que foca na utilização da aplicação pelo usuário, se o layout está correto, se a interface está de acordo com as necessidades do cliente, entre outros.
- *Smoke test*: É o teste que verifica os pontos mais críticos do *software*, é executado após a aplicação estar pronta no ciclo de desenvolvimento, para ser encaminhada para o ciclo de testes.

2.4 AUTOMAÇÃO DE TESTES

Conforme Sommerville (2011, p. 147), em testes automatizados, os casos de testes e seus cenários são todos codificados no programa que a empresa estiver utilizando e serão realizados toda vez que for feita uma mudança na aplicação.

Sommerville (2011, p. 147) declara que recentemente o uso de testes automatizados vem aumentando cada vez mais no mercado, devido ao tempo que as equipes estão economizando ao invés de gastarem muitas horas testando o software, a automação realiza os testes e a equipe poderá focar no desenvolvimento de outra funcionalidade ou o aprimoramento de uma já existente.

Contudo Sommerville (2011, p. 147) declara que os testes realizados nunca serão totalmente automatizados, visto que a automação apenas valida se a aplicação possui o comportamento que é solicitado, testes que são necessários validar a interface da aplicação e seu layout não são possíveis de automatizar.

> Testes automatizados são programas ou scripts simples que exercitam funcionalidades do sistema sendo testado e fazem verificações automáticas nos efeitos colaterais obtidos. A grande vantagem desta abordagem, é que todos os casos de teste podem ser facilmente e rapidamente repetidos a qualquer momento e com pouco esforço. (BERNARDO; KON, 2008)

De acordo com Bernardo e Kon (2008) a automação possibilita a criação de testes manuais mais elaborados e complexos para serem executados pela equipe a qualquer instante durante o projeto e transmite confiança para a equipe de que podem ser realizadas alterações no código do software.

2.5 BDD

O BDD (*Behavior Driven Development*, em português, desenvolvimento orientado por comportamento) segundo Dan North (2003, apud CAROLI, 2014, p. 244) é uma evolução do TDD (*Test Driven Development*, em português, desenvolvimento orientado por testes) criada

com o intuito de expor de maneira mais clara a forma correta de se utilizar o TDD, pois muitos desenvolvedores permaneciam confusos quanto a maneira de testar, o que testar e quando testar.

Com isso o BDD possui os seguintes fundamentos de acordo com Caroli (2014, p. 244):

- Um fluxo de trabalho mais bem definido, através de uma abordagem outside-in (“de fora para dentro”, em uma tradução livre)
- Uma clara compreensão das funcionalidades por ambas as partes, isto é, pela equipe técnica e pelos *stakeholders*. Para tanto, usa-se linguagem natural e procura-se ser o mais descritivo possível no momento da escrita de cenários de teste e de seus respectivos exemplos, uma vez que estes são, na realidade, especificações de um comportamento desejado, sendo que todos devem ser capazes de contribuir para o seu aprimoramento.
- Funcionalidades que possuam um valor claro e verificável para o negócio, de modo a priorizar o mais importante e evitar o que pode ser desnecessário.
- Reforço da importância do uso de exemplos concretos para descrever uma funcionalidade, seja ela uma especificação de baixo ou de alto nível.
- Um mínimo planejamento futuro, escrevendo especificações para o menor e mais prioritário subconjunto de funcionalidades possível, desenvolvendo-o antes de adicionar mais funcionalidades.

Em suma conforme Caroli (2014, p. 245) o BDD possui como principal objetivo aumentar a eficácia da equipe através do aperfeiçoamento da comunicação. De maneira que quaisquer que sejam as funções do colaborador, analista de negócios, desenvolvedor, ou analista de qualidade, ele consiga entender o que estiver descrito na documentação a respeito das funcionalidades da aplicação

2.6 FERRAMENTAS DE AUTOMAÇÃO DE TESTES

Segundo Ferreira (2019) a automação de testes pode ser dividida em 3 áreas que compõem a Pirâmide de Automação, exibida abaixo.

Figura 5 – Pirâmide de Automação de Testes

Fonte: Ferreira (2019)

A seguir são explicadas cada área apresentada na pirâmide sua respectiva importância, e quais ferramentas podem ser utilizadas em cada uma delas.

2.6.1 Testes Unitários

Há inúmeros testes unitários, pois segundo Ferreira (2019) são facilmente construídos e o custo para permanecerem atualizados e funcionando é baixo. Testes unitários possuem diferentes conceitos dependendo da empresa, certas empresas definem como o software completo, outras como um módulo ou um serviço.

Existem muitas ferramentas para realizar a automação deste tipo de teste, com base na linguagem utilizada na programação, a seguir Ferreira (2019) apresenta uma lista que contém as ferramentas básicas para automatizar os testes de unitários.

- JUNit: Ferramenta utilizada quando a linguagem usada for Java
- Visual Studio Unit Testing Framework: Ferramenta utilizada quando a linguagem usada for .Net
- Unit.js: Ferramenta utilizada quando a linguagem usada for Javascript
- PHPUnit: Ferramenta utilizada quando a linguagem usada for PHP
- PyUnit (unittest): Ferramenta utilizada quando a linguagem usada for Python

2.6.2 Testes de integração

Ferreira (2019) afirma que testes de integração são os testes que atravessam as unidades do código-fonte, são testes mais caros para manterem atualizados e funcionais, devido a dependência entre um componente e outro ou até mesmo mais componentes entre si, antes de se realizar uma alteração em um componente específico deve ser levado em conta se esta mesma alteração irá impactar no teste de outro componente relacionado.

As ferramentas listadas por Ferreira (2019) para este tipo de teste, são as mesmas listadas anteriormente com a adição de 2 ferramentas.

- Postman: Ferramenta utilizada para integração com APIs[1]
- DB Unit: Ferramenta utilizada para integração com banco de dados

[1] API: Application Programming Interface/Interface de Programação de Aplicativos

2.6.3 Testes de Interface & Aceitação

Segundo Ferreira (2019) testes de interface e aceitação são os mais difíceis de serem criados e os mais caros a serem mantidos atualizados e funcionais, devido o envolvimento de interfaces gráficas que possuem grandes chances de serem modificadas conforme o andar do projeto. Fazem parte dos testes caixa-preta apresentados anteriormente.

Ferreira (2019) lista as seguintes ferramentas para estes tipos de teste

- Selenium: Ferramenta utilizada para automação em ambientes web
- Espresso: Ferramenta utilizada para automação em ambiente Android
- Xcode UI Tests: Ferramenta utilizada para automação em ambiente iOS
- Visual Studio Coded UI Test: Ferramenta utilizada para automação em ambiente Windows
- Cucumber: Ferramenta utilizada quando a linguagem for Java
- SpecFlow: Ferramenta utilizada quando a linguagem for .Net
- Jasmine: Ferramenta utilizada quando a linguagem for JavaScript
- PHPSpec: Ferramenta utilizada quando a linguagem for PHP
- Behave: Ferramenta utilizada quando a linguagem for Python

2.6.4 Ferramentas de automação mais utilizadas

Diogo (2019) lista, de maneira objetiva, as ferramentas de automação de testes mais usadas atualmente, e são apresentadas abaixo.

- Appium: Ferramenta utilizada para serem realizados testes funcionais para sistemas operacionais *Android* e iOS
- Selenium: Ferramenta que permite múltiplas maneiras de automação, mas a mais usada é em ambiente web através da gravação e reprodução
- Robotium: Ferramenta utilizada para realizar testes de interface gráfica em dispositivos *Android*

- Katalon Studio: Ferramenta gratuita baseada no Selenium e Appium utilizada para serem realizadas automações em ambientes web e dispositivos móveis
- Cucumber: Ferramenta utilizada para testes de aceitação para aplicações web, orientada por comportamento (BDD)
- Telerik Test Studio: Ferramenta utilizada para automação de testes de performance, de carregamento, e exploratórios.

2.6.5 Selenium

O Selenium é uma ferramenta *open source* focada na criação de testes automatizados funcionais para aplicações que executam sob protocolos *web* como HTTP ou HTTPS. Com a utilização da ferramenta é possível gerar *scripts* de teste, por meio da técnica *Record&Playback.* Além disso, o Selenium é suportado por diversos navegadores web, como por exemplo Mozilla Firefox e Internet Explorer.

"Outra importante característica desta ferramenta, está relacionada à sua compatibilidade com diversas plataformas. Pelo fato de ser desenvolvido na linguagem Java, pode ser executado em sistemas operacionais Windows, Linux e MacOS". (COSTA, 2012, p. 104)

Sua arquitetura é composta por três ferramentas, com as quais é possível criar e executar casos de teste:

- Selenium-IDE
- Selenium-RC (*Remote Control*)
- Selenium-Grid

2.6.5.1 Selenium-IDE

Segundo Costa (2012, p. 104), o Selenium-IDE é um ambiente de desenvolvimento do Selenium e a partir dele é realizada a criação de casos de teste e suítes de teste. Ele funciona como uma extensão do browser e possui um recurso de gravação que registra cada interação do usuário com a aplicação, citado anteriormente como *Record&Playback*. Posteriormente, cada interação registrada é utilizada para a geração dos scripts de teste.

Para um público iniciante e com pouca experiência, é recomendado o uso do Selenium-IDE, devido sua facilidade de trabalho e de automação de testes, uma vez que não é necessário um conhecimento prévio em programação para utilizá-lo.

Entretanto, a criação de alguns casos de teste para o Selenium-IDE pode ser relativamente complexa, principalmente quando se faz necessária a utilização de lógica de programação.

2.6.5.2 Selenium-RC

Costa (2012, p. 104) apresenta o Selenium-RC como uma ferramenta que permite a execução de casos de teste e, é a recomendada para a geração de casos de teste mais complexos. Esta ferramenta é utilizada quando se faz necessária uma análise mais profunda e criteriosa dos casos de teste ou até mesmo a realização de consultas em banco de dados ou outros sistemas integrados.

2.6.5.3 Selenium-Grid

O Selenium-Grid permite maior escalabilidade dos testes, uma vez que possibilita a execução simultânea de múltiplos Selenium-RC. Com o Selenium-Grid, várias instâncias do

Selenium-RC são executadas em paralelo de forma a simular um ambiente de teste mais próximo do real. (COSTA, 2012, p. 104)

3 METODOLOGIA

Neste capítulo, é abordado a caracterização do tipo de pesquisa, as etapas metodológicas, bem como o planejamento de atividades, a proposta de solução e suas delimitações

3.1 CARACTERIZAÇÃO DO TIPO DE PESQUISA

De acordo com Silva e Menezes (2001, p. 20) “pesquisa é um conjunto de ações, propostas para encontrar a solução para um problema, que têm por base procedimentos racionais e sistemáticos”.

Continuando Silva e Menezes (2001, p. 20) afirmam que pesquisas são realizadas somente quando há algum tipo de problema e não existem informações de como solucioná-lo.

Há inúmeras maneiras de se classificar pesquisas, Silva e Menezes (2001, p.20) apresentam as formas clássicas de acordo com o ponto de vista da natureza da pesquisa e as descrevem a seguir:

- Pesquisa Básica: objetiva gerar conhecimentos novos úteis para o avanço da ciência sem aplicação prática prevista. Envolve verdades e interesses universais.
- Pesquisa Aplicada: objetiva gerar conhecimentos para aplicação prática e dirigidos à solução de problemas específicos. Envolve verdades e interesses locais.

Neste trabalho é utilizado a pesquisa aplicada, pois serão propostas novas metodologias de testes de software, técnicas e estratégias com o intuito de auxiliar o processo de testes, desenvolvimento e análise de software na empresa estudada.

Dentro destas metodologias serão utilizadas técnicas de desenvolvimento de *software* como por exemplo TDD e BDD, além de técnicas serão utilizadas ferramentas como o *Cucumber*, que é uma ferramenta usada para ler os cenários de testes, criados pelo *tester*, e

executá-los de forma automatizada, para facilitar o entendimento dos testes de *software* realizados para o desenvolvedor

Segundo Silva e Menezes (2001, p. 20) as classificações de pesquisas do ponto de vista da forma de abordagem do problema podem ser:

- Pesquisa Quantitativa: considera que tudo pode ser quantificável, o que significa traduzir em números opiniões e informações para classificá-las e analisá-las. Requer o uso de recursos e de técnicas estatísticas (percentagem, média, moda, mediana, desvio-padrão, coeficiente de correlação, análise de regressão, etc.).
- Pesquisa Qualitativa: considera que há uma relação dinâmica entre o mundo real e o sujeito, isto é, um vínculo indissociável entre o mundo objetivo e a subjetividade do sujeito que não pode ser traduzido em números. A interpretação dos fenômenos e a atribuição de significados são básicas no processo de pesquisa qualitativa. Não requer o uso de métodos e técnicas estatísticas. O ambiente natural é a fonte direta para coleta de dados e o pesquisador é o instrumento-chave. É descritiva. Os pesquisadores tendem a analisar seus dados indutivamente. O processo e seu significado são os focos principais de abordagem.

Neste trabalho a pesquisa realizada é classificada do ponto de vista da forma de abordagem como qualitativa.

Do ponto de vista de objetivos, Gil (1991, apud SILVA; MENEZES, 2001, p. 21) caracteriza a pesquisa em:

- Pesquisa Exploratória: visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses. Envolve levantamento bibliográfico; entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado; análise de exemplos que estimulem a compreensão. Assume, em geral, as formas de Pesquisas Bibliográficas e Estudos de Caso.

- Pesquisa Descritiva: visa descrever as características de determinada população ou fenômeno ou o estabelecimento de relações entre variáveis. Envolve o uso de técnicas padronizadas de coleta de dados: questionário e observação sistemática. Assume, em geral, a forma de Levantamento.
- Pesquisa Explicativa: visa identificar os fatores que determinam ou contribuem para a ocorrência dos fenômenos. Aprofunda o conhecimento da realidade porque explica a razão, o "porquê" das coisas. Quando realizada nas ciências naturais, requer o uso do método experimental, e nas ciências sociais requer o uso do método observacional. Assume, em geral, a formas de Pesquisa Experimental e Pesquisa Expost-facto.

Neste trabalho a pesquisa realizada é classificada do ponto de vista dos objetivos como exploratória.

3.2 ETAPAS METODOLÓGICAS

Para atingir os objetivos deste trabalho as seguintes atividades devem ser desenvolvidas:

- Pesquisa da literatura sobre qualidade de software, testes, automação de testes.
- Pesquisa de tecnologias e ferramentas para automação de testes
- Descrição da empresa
- Descrição do atual processo de desenvolvimento e de testes da empresa estudo de caso
- Levantamento dos subprocessos ou atividades de teste mais aptos para automação
- Escolha de ferramentas e tecnologias
- Aplicação piloto de automação de testes
- Definição do processo de testes automatizados
- Apresentação para a empresa da sugestão de processo de testes automatizado.

3.3 DELIMITAÇÕES

No desenvolvimento deste trabalho são apresentadas as delimitações da pesquisa a seguir:

- A proposta de automação de testes será implantada somente no projeto escolhido pela empresa.
- Não necessariamente a proposta do modelo de desenvolvimento será acolhida e adotada pela equipe.

4 DESENVOLVIMENTO DA PROPOSTA

Neste capítulo são descritas características da empresa, equipe, processo de desenvolvimento e propostas para melhorá-lo.

4.1 DESCRIÇÃO DA EMPRESA

A empresa estudada iniciou no mercado como uma empresa fábrica de software terceirizada, ou seja, desenvolvia aplicações, sites, funcionalidades, apps conforme as solicitações do cliente. Entretanto com a grande demanda do mercado e a velocidade que novas tecnologias vinham surgindo a empresa viu a necessidade de uma transformação, isso no final do ano de 2017.

Com isso, a empresa através de alianças, inovações no ambiente interno e na maneira de gerenciamento, foco na participação de eventos e exposição da marca, alcançou o necessário para continuar no mercado, ao invés de ser mais uma empresa fábrica de software, se tornou uma empresa de comunicação digital.

A empresa é composta por um total de 37 colaboradores, divididos em 3 núcleos de TI. Um dos núcleos é responsável pela parte web e sistemas legados. Esse núcleo, integrado por 14 pessoas, é dividido em uma equipe de qualidade composta por 2 colaboradores, uma equipe de desenvolvedores composta por 4 colaboradores, uma equipe de análise composta por 3, uma equipe de marketing composta por 4 e um gestor.

O segundo núcleo é responsável pelo desenvolvimento da parte mobile da empresa e é composto por um único analista de qualidade, 3 desenvolvedores, 2 analistas de TI, e um gestor.

O terceiro núcleo é responsável pela parte de infraestrutura composto por um gestor e 3 colaboradores. Além dos núcleos de TI a empresa possui o setor de RH composto por 2 colaboradores.

A seguir é apresentado o processo de desenvolvimento e de testes da equipe de desenvolvimento.

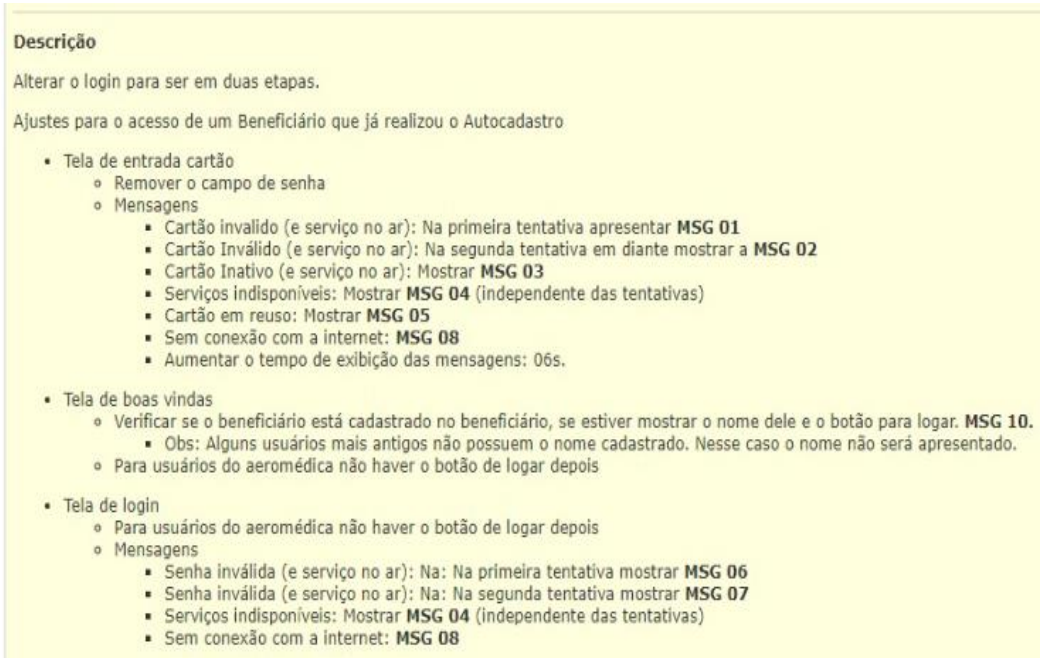
4.2 PROCESSO DE DESENVOLVIMENTO

O processo de desenvolvimento de uma nova aplicação, funcionalidade ou site na empresa é caracterizado pela chegada de uma nova demanda do cliente, através da criação de um chamado na plataforma *ServiceDesk*, solicitando uma melhoria ou reportando um defeito encontrado pelo próprio. A demanda é analisada e estudada pelos analistas de T.I e pelo gestor em uma reunião fechada, com o intuito de verificar se é válida a inserção da demanda no decorrer do projeto. É acordado entre os analistas e cliente uma data estimada para a conclusão do desenvolvimento da demanda e sua implantação no ambiente de produção.

Ao concluir esta etapa, é gerada uma documentação com todas as regras de negócios, requisitos, solicitações específicas do cliente, dados para as simulações de testes, e a tarefa da demanda propriamente dita para ser assumida pela equipe do projeto.

Tal documentação é apresentada abaixo como figura:

Figura 6 - Documentação de tarefa



Descrição

Alterar o login para ser em duas etapas.

Ajustes para o acesso de um Beneficiário que já realizou o Autocadastro

- Tela de entrada cartão
  - Remover o campo de senha
  - Mensagens
    - Cartão invalido (e serviço no ar): Na primeira tentativa apresentar **MSG 01**
    - Cartão Inválido (e serviço no ar): Na segunda tentativa em diante mostrar a **MSG 02**
    - Cartão Inativo (e serviço no ar): Mostrar **MSG 03**
    - Serviços indisponíveis: Mostrar **MSG 04** (independente das tentativas)
    - Cartão em reuso: Mostrar **MSG 05**
    - Sem conexão com a internet: **MSG 08**
    - Aumentar o tempo de exibição das mensagens: 06s.
- Tela de boas vindas
  - Verificar se o beneficiário está cadastrado no beneficiário, se estiver mostrar o nome dele e o botão para logar. **MSG 10.**
    - Obs: Alguns usuários mais antigos não possuem o nome cadastrado. Nesse caso o nome não será apresentado.
  - Para usuários do aeromédica não haver o botão de logar depois
- Tela de login
  - Para usuários do aeromédica não haver o botão de logar depois
  - Mensagens
    - Senha inválida (e serviço no ar): Na: Na primeira tentativa mostrar **MSG 06**
    - Senha inválida (e serviço no ar): Na: Na segunda tentativa mostrar **MSG 07**
    - Serviços indisponíveis: Mostrar **MSG 04** (independente das tentativas)
    - Sem conexão com a internet: **MSG 08**

Fonte: Autor, 2019

Essa documentação é disponibilizada pelos analistas para a equipe do projeto através da criação da tarefa, relacionada a demanda exigida, no ambiente do *ServiceDesk* com

o status “Aguardando desenvolvimento” junto com uma breve descrição do que deve ser feito e em qual projeto ela se encaixa, *web* ou *mobile*.

Com isso, a equipe de desenvolvimento inicia seu processo para realizar andamento da atividade alterando o status da tarefa de “Aguardando desenvolvimento” para “Em execução”, desenvolvendo a funcionalidade seguindo o que foi escrito pelo analista na documentação e na breve descrição da tarefa, realizando testes unitários e testes de caixa-branca com o intuito de verificar os possíveis erros de código.

Finalizando o desenvolvimento da tarefa e os respectivos testes descritos, o desenvolvedor responsável altera o status da tarefa para “Desenvolvimento Finalizado” conforme a figura abaixo indica:

Figura 7 - Status de tarefa

Fonte: Autor, 2019

Após isso a tarefa é implantada no ambiente de testes, pelo próprio desenvolvedor da tarefa, seu status é alterado para “Aguardando Teste”, e é liberada para equipe de testes.

A equipe de testes estuda a documentação, cria sua tarefa de teste e respectivos casos de aceite para cada funcionalidade prestes a ser testada. O status da tarefa é alterado para “Em teste”, os testes se iniciam dos mais básicos, como validação visual da tarefa, se está de acordo com o que foi descrito e acordado, se a mesma é exibida corretamente em múltiplos navegadores, e avançam para testes caixa preta, testes de funcionalidade, de usabilidade, de regressão, entre outros. Se o *tester* julgar necessário e a funcionalidade requisitar, testa o

comportamento da aplicação em diferentes dispositivos, se for abordado a questão de dispositivos móveis.

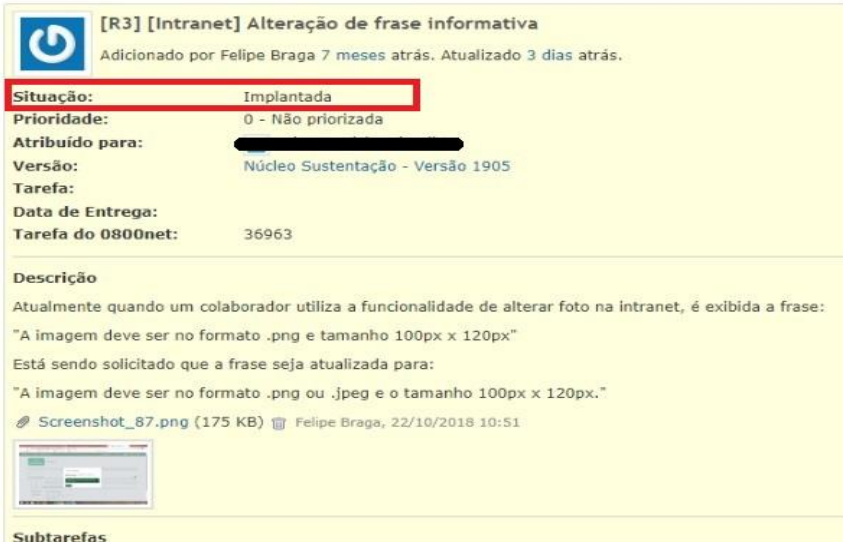
Após realizar todo o processo de verificação da aplicação, e caso não forem encontrados erros visuais ou funcionais, como por exemplo um botão não realizando a função especificada pelo analista ou a visualização da página não estiver corretamente alinhada ao acessar utilizando o navegador *Internet Explorer*, a tarefa então é aprovada pela equipe de testes, seu status é alterado para “Teste Aprovado”, e é repassada novamente para os analistas.

Estes realizam a homologação da tarefa, que é descrita como a verificação da aplicação assumindo o papel de um usuário, ou seja, simulando apenas o caminho feliz da aplicação, utilizando as funcionalidades de forma direta e objetiva, como por exemplo utilizando a busca da página para ser encontrado um médico em sua região.

Na homologação não são realizados cenários de falha, como por exemplo realizar a mesma busca por um médico, porém com o campo não preenchido, pois a validação realizada é para verificar se a aplicação é funcional e sua visualização está de acordo com as demandas do cliente.

Se for aprovada pelos analistas, o status da tarefa é modificado para “Aprovada” e é repassada novamente para os desenvolvedores. Estes realizam implantação das tarefas três dias antes do que foi acordado entre os analistas e o cliente no ambiente de produção, a implantação é feita dias antes por questões de diminuição de riscos, caso ocorra algum problema durante a execução da mesma, com este passo concluído o status da tarefa passa a ser “Implantada”. A figura a seguir apresenta esse cenário.

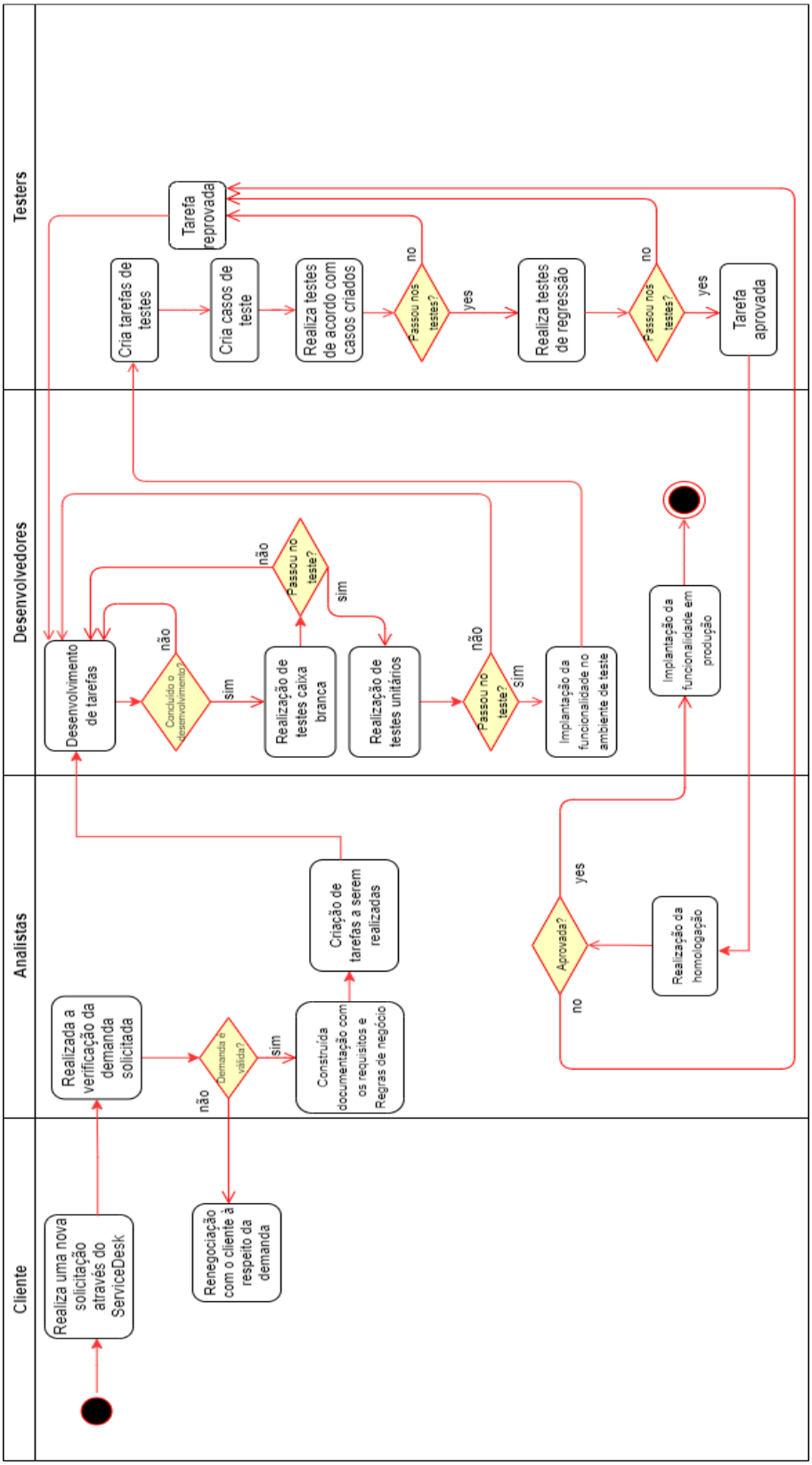
Figura 8 - Tarefa implantada



Fonte: Autor, 2019

Todo o processo descrito é representado no diagrama de processos a seguir:

Figura 9 - Processo de desenvolvimento atual



Fonte: Autor, 2019

4.3 PROPOSTA DE NOVO PROCESSO DE DESENVOLVIMENTO

Concomitante à proposta de um novo processo de desenvolvimento, certos fragmentos do processo atual terão que sofrer certas alterações, além disso serão adicionados novos passos e reuniões respeitando a metodologia escolhida que é o Scrum.

Após uma nova solicitação de demanda ser feita pelo cliente, por meio da plataforma utilizada na empresa, o *ServiceDesk*, essa mesma demanda será analisada pelos analistas de T.I com o intuito de esclarecer para o cliente se a mesma é válida ou não para ser inserida na próxima *Sprint*, que é um limite de tempo específico de trabalho, o qual tem uma duração média de 2 a 4 semanas.

Junto com a demanda analisada e verificada, será realizada uma reunião entre os analistas e o *tester*, nela serão esclarecidos os pontos principais da demanda e será criada a documentação, com regras de negócio e termos de aceite acordados, chamado de *Product Backlog*.

Com a finalização da reunião, os analistas criarão as histórias a serem realizadas no decorrer da *sprint* pela equipe do projeto, time scrum, e o *tester* irá iniciar a criação de cenários de teste para estas tarefas.

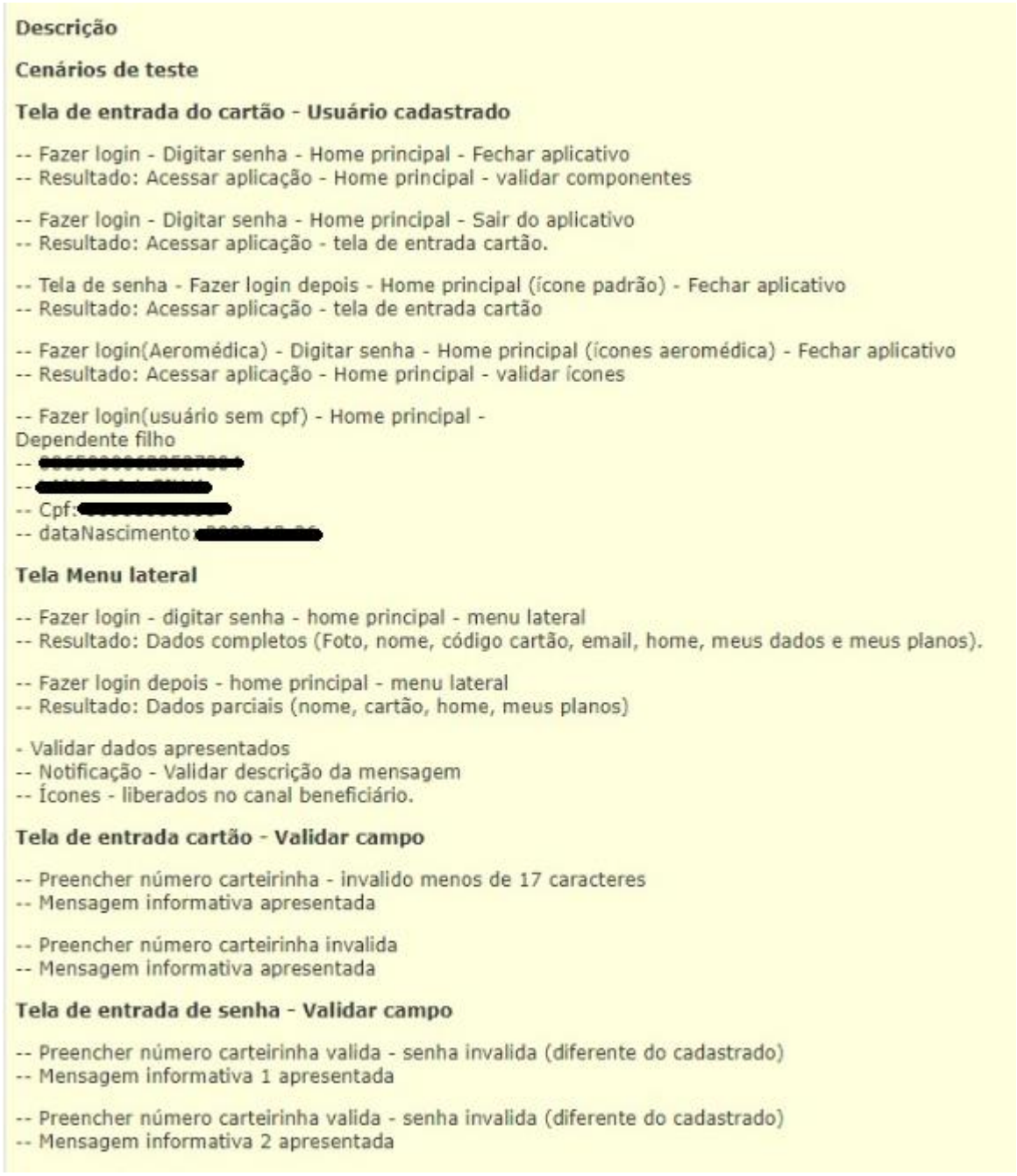
Feitas as definições do que será realizado na *sprint* e os determinados cenários de teste a serem levados em consideração, será feita uma reunião entre todo o time scrum que é chamada de planejamento da *Sprint*. Nessa reunião é compartilhado com todos os membros do time as documentações de cada tarefa e são definidas as prioridades de cada uma, levando em consideração fatores como: tempo necessário, complexidade, risco, entre outros.

As tarefas então passam a estarem com o status de “Aguardando desenvolvimento” até que algum desenvolvedor do projeto decida iniciá-la.

Com isso se iniciará o período de desenvolvimento da *sprint*. Nessa etapa os desenvolvedores do projeto começarão a desenvolver as tarefas que foram inseridas na *sprint*, sendo guiados pela documentação criada pelos analistas e pelos comportamentos definidos nos cenários criados pelo tester, aplicando corretamente a prática do BDD no projeto. Dessa maneira com uma linguagem natural e de fácil entendimento, os desenvolvedores conseguem realizar suas tarefas com a visão dos analistas e do *tester* em relação a como a nova funcionalidade, site ou aplicação deve se comportar por exemplo em um cenário de falha.

A seguir, um exemplo de cenário criado pelo tester

Figura 10 – Cenário de Teste



Descrição

Cenários de teste

Tela de entrada do cartão - Usuário cadastrado

-- Fazer login - Digitar senha - Home principal - Fechar aplicativo
-- Resultado: Acessar aplicação - Home principal - validar componentes

-- Fazer login - Digitar senha - Home principal - Sair do aplicativo
-- Resultado: Acessar aplicação - tela de entrada cartão.

-- Tela de senha - Fazer login depois - Home principal (ícone padrão) - Fechar aplicativo
-- Resultado: Acessar aplicação - tela de entrada cartão

-- Fazer login(Aeromédica) - Digitar senha - Home principal (ícones aeromédica) - Fechar aplicativo
-- Resultado: Acessar aplicação - Home principal - validar ícones

-- Fazer login(usuário sem cpf) - Home principal -
Dependente filho
--
--
-- Cpf:
-- dataNascimento

Tela Menu lateral

-- Fazer login - digitar senha - home principal - menu lateral
-- Resultado: Dados completos (Foto, nome, código cartão, email, home, meus dados e meus planos).

-- Fazer login depois - home principal - menu lateral
-- Resultado: Dados parciais (nome, cartão, home, meus planos)

- Validar dados apresentados
-- Notificação - Validar descrição da mensagem
-- Ícones - liberados no canal beneficiário.

Tela de entrada cartão - Validar campo

-- Preencher número carteirinha - invalido menos de 17 caracteres
-- Mensagem informativa apresentada

-- Preencher número carteirinha invalida
-- Mensagem informativa apresentada

Tela de entrada de senha - Validar campo

-- Preencher número carteirinha valida - senha invalida (diferente do cadastrado)
-- Mensagem informativa 1 apresentada

-- Preencher número carteirinha valida - senha invalida (diferente do cadastrado)
-- Mensagem informativa 2 apresentada

Fonte: Autor, 2019

Se determinada funcionalidade não realizar com sucesso o cenário listado pelo tester, ela continuará com o status “Em desenvolvimento” e deverá ser corrigida pelo desenvolvedor responsável pela mesma.

A tarefa desenvolvida com sucesso e aprovada nos cenários de testes criados pelo *tester*, será implantada no ambiente de testes pelo desenvolvedor, terá seu status alterado para "Aguardando teste" e será atribuída para o *tester*.

Após ser implantada com sucesso, e o tester assumir a tarefa, seu status será alterado para "Em teste", o *tester* criará sua respectiva tarefa de teste para trabalhar, e dará início ao período de testes desta tarefa em questão.

Os testes realizados pelo *tester* serão focados em verificar possibilidades não listadas nos cenários criados anteriormente, desde um teste que não poder ser realizado por código até possibilidades não previstas nos cenários, mas que podem vir a ocorrer futuramente.

Não há uma lista de testes específica para cada funcionalidade, os testes irão variar para cada tipo de tarefa, como por exemplo validações visuais para verificar se as cores estão de acordo com o escopo do projeto, se a exibição está correta em múltiplos navegadores e em dispositivos móveis, se está sendo verificado se algum campo está vazio ou não, entre outros.

Com o processo de testes finalizado, e se a tarefa não apresentar erros visuais e/ou funcionais, como por exemplo um campo não sendo validado se está preenchido ou não, ou a visualização da página não estiver semelhante ao escopo ao acessar utilizando um dispositivo móvel, a mesma será aprovada pelo *tester*, terá seu status alterado "Teste Aprovado".

O *tester* então retorna a tarefa para os desenvolvedores, que irão implantar a funcionalidade aprovada no ambiente de produção. A tarefa poderá ser verificada novamente, se os analistas julgarem necessário, para ter certeza que não há diferenças entre o ambiente de testes e o ambiente de produção.

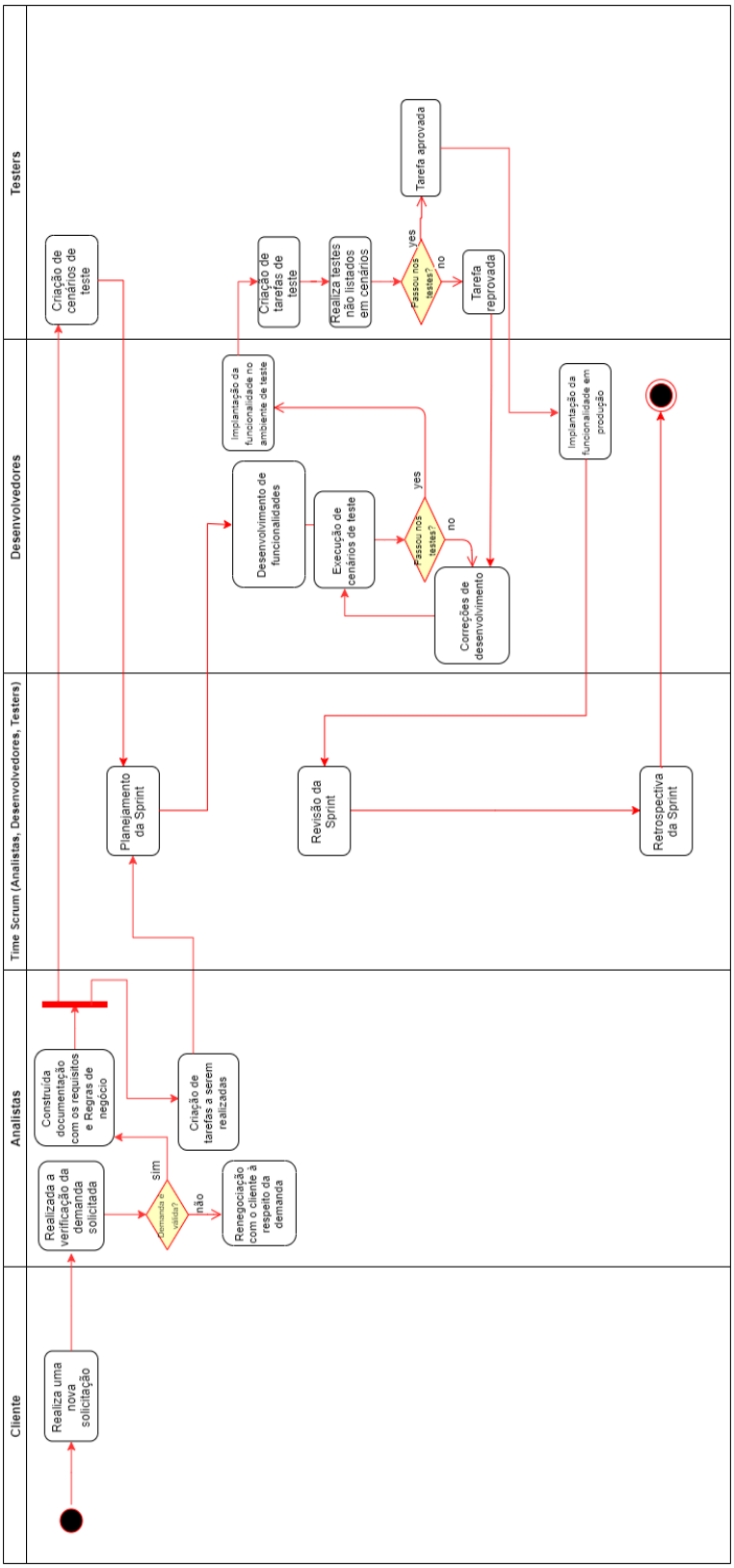
Este processo será feito em todas as funcionalidades e tarefas que foram acordadas para serem terminadas na *sprint*, até o final do período acordado de 1 mês, e com suas devidas conclusões, e aprovações, serão realizadas duas reuniões pelo time inteiro, a *sprint review* e *sprint retrospective*.

A *sprint review* será uma reunião de curta duração, no máximo 2 horas, onde será abordado e revisado se todas as tarefas acordadas na reunião de planejamento foram concluídas, se alguma tarefa reprovada será abordada em uma próxima *sprint*, entre outros.

A *sprint retropective* é a reunião onde time scrum avalia o que foi considerado bom na *sprint* e o que deve ser melhorado pelo time, como por exemplo se a comunicação entre a equipe foi vantajosa para o andamento do projeto, ou se houve atraso na alteração de status pelos desenvolvedores. Com isso, tomando precauções para que o time continue como está ou tome ações para melhorar o desenvolvimento futuro.

Todo este novo processo de desenvolvimento descrito é representado no diagrama a seguir:

Figura 11 - Nova proposta de processo de desenvolvimento



Fonte: Autor, 2019

4.4 PROPOSTA DE AUTOMAÇÃO DE TESTES

As ferramentas escolhidas para serem utilizadas na automação dos testes no decorrer da sprint são apresentadas abaixo, além delas é utilizada a linguagem de programação Ruby e a técnica de desenvolvimento guiado por comportamento, o BDD.

4.4.1 Ruby

"A linguagem Ruby foi criada por Yukihiro Matsumoto, mais conhecido como Matz, no ano de 1995 no Japão, com o objetivo de ser uma linguagem mais legível e agradável de se programar." (SOUZA, 2014, p. 1)

Conforme Souza (2014, p. 11) uma característica importante da linguagem que a diferencia das demais é o fato de o Ruby ser uma linguagem que possui tipagem forte e dinâmica, em que variáveis podem ser atribuídas dinamicamente, permitindo que seu tipo possa ser alterado durante a execução do programa.

Segundo Souza (2014, p. 13) a linguagem é frequentemente usada para a criação de *scripts* para a leitura e processamento de arquivos, automação de *builds* e *deploys*, realizar a coleta de informações de códigos de sites para serem analisados, entre outros.

Ainda assim, Souza (2014, p. 13) afirma que o que mais alavancou a linguagem no mercado foram suas aplicações web e a criação do *Ruby on Rails*.

> Ele é um framework web que segue o padrão MVC (ModelView-Controller) e evangeliza o Convention over configuration, ou seja, acabar com aquela quantidade enorme de XMLs que existem na maioria dos frameworks web que existem no mercado e que dificultam muito a manutenção. (SOUZA, 2014, p. 13)

O mercado em torno da linguagem Ruby é gigantesco atualmente, e a tendência é que continue crescendo ao passar do tempo. Vários serviços estão sendo criados e uma grande quantidade de Startups apostam na linguagem e em suas aplicações. (SOUZA, 2014, p. 14)

Baggio (2012, p. 6) afirma que o Ruby possui um gerenciador de pacotes muito flexível, dinâmico, e de fácil entendimento denominado “RubyGems”.

As gems podem ser vistas como bibliotecas, conjuntos de arquivos em código Ruby reutilizáveis pela comunidade. Nessas bibliotecas podem ser encontrados, por exemplo plugins para relatórios, layouts, ferramentas como o Cucumber, entre outros.

4.4.2 Cucumber

Segundo Wynne (2017, p. 17/Preface XXVII) o *Cucumber* é uma ferramenta de teste de aceitação orientada pela metodologia BDD, desenvolvida com o propósito de descrever o valor do negócio em uma linguagem de fácil entendimento, entre os membros técnicos e não técnicos de uma equipe, a respeito das demandas do projeto através de descrições funcionais, conhecidas como cenários, das solicitações do cliente.

Seguindo Wynne (2017, p. 26/7) as especificações dos cenários são realizadas em um arquivo texto, onde são descritas as pré-condições, passos e resultados esperados através de uma linguagem natural, deixando de lado de qualquer termo técnico para facilitar o entendimento. A partir da execução desse arquivo, será verificado e validado o comportamento do software de acordo com a expectativa do usuário.

Estes arquivos de texto são limitados a um formato específico de linguagem/regras de sintaxe básica, conhecida como *Gherkin*.

4.4.3 Sintaxe Gherkin

Conforme Chaim (2018) Gherkin é uma linguagem natural de fácil entendimento com o intuito de fazer com que tanto o cucumber quanto todos os membros do time, inclusive o cliente, entendam o que foi acordado e planejado para ser realizado no decorrer do projeto, gerando uma documentação viva e de valor pro negócio.

O Gherkin possui palavras chave para que seja possível se comunicar com o cucumber, e estas são:

- Dado (Given): O “Dado” representa as pré-condições do cenário.
- Quando (When): O “Quando” representa as ações que o usuário executa no sistema/aplicação durante o cenário.
- Então (Then): O “Então” representa as saídas, os resultados esperados do cenário da aplicação.

Exemplo de documentação com sintaxe gherkin:

Figura 12 – Documento com sintaxe Gherkin

```
Funcionalidade:Usuário Login
    Para que eu possa cadastrar e gerenciar minhas tarefas
    Sendo um usuário
    Posso acessar o sistema com meu email e senha previamente cadastrados

    Fazer login com email e senha
    Clicou em OK, é autenticado com sucesso
    Olá, Fulano

    Cenario: Usuário deve ser autorizado

        Dado que eu acesso a página principal
        Quando eu faço login com "exemplo@teste.com" e "123456"
        Então devo ser autenticado com sucesso
        E devo ver a seguinte mensagem "Olá, Exemplo"

    Cenario: Senha errada

        Dado que eu acesso a página principal
        Quando eu faço login com "exemplo@teste.com" e "654321"
        Então devo ver a seguinte mensagem "Senha inválida"

    Cenario: Usuário não existe

        Dado que eu acesso a página principal
        Quando eu faço login com "exemplo@teste.net" e "654321"
        Então devo ver a seguinte mensagem "Usuário não cadastrado."
```

Fonte: Autor, 2019

Além destas palavras chave, Gherkin possui uma opção para reduzir a reprodução de uma mesma afirmação que poderia se repetir em vários cenários e com isso gerar retrabalho, neste caso em específico a afirmação que se repete é “Dado que eu acesso a página principal”.

Esta opção é chamada de “Contexto”, onde ela é executada sempre antes de qualquer um dos cenários criados, como uma pré-condição. A figura a seguir apresenta o exemplo utilizando a opção “contexto”.

Figura 13 – Sintaxe Gherkin com contexto

```
Contexto: Página Principal
    Dado que eu acesso a página principal

Cenario: Usuário deve ser autorizado

    Quando eu faço login com "exemplo@teste.com" e "123456"
    Então devo ser autenticado com sucesso
    E devo ver a seguinte mensagem "Olá, Exemplo"

Cenario: Senha errada

    Quando eu faço login com "exemplo@teste.com" e "654321"
    Então devo ver a seguinte mensagem "Senha inválida"

Cenario: Usuário não existe

    Quando eu faço login com "exemplo@teste.net" e "654321"
    Então devo ver a seguinte mensagem "Usuário não cadastrado."

Cenario: Email incorreto

    Quando eu faço login com "exemplo.net" e "123456"
    Então devo ver a seguinte mensagem "Email incorreto ou ausente."
```

Fonte: Autor, 2019

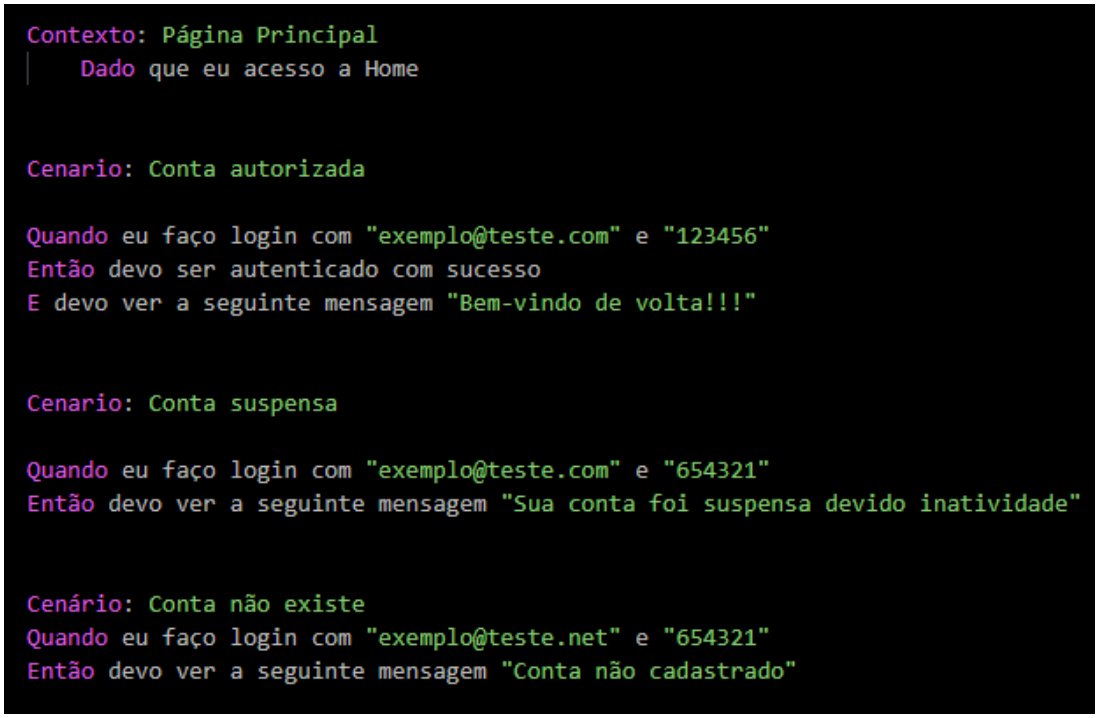
4.4.4 Automação dos testes

A proposta de automação do novo processo de desenvolvimento inicia logo após ocorrer a reunião entre o *tester* e os analistas, e após os principais pontos da demanda, como suas regras de negócio e termos de aceite serem acordados. A primeira etapa da automação consiste na criação de cenários de teste pelo *tester*, utilizando a linguagem Gherkin.

O uso dessa linguagem tem o intuito de guiar os desenvolvedores mostrando qual o comportamento esperado para cada situação específica da tarefa, como por exemplo qual o

comportamento da página quando certo campo permanecer em branco após o usuário tentar enviar um formulário.

Figura 14 – Cenário de teste com Gherkin



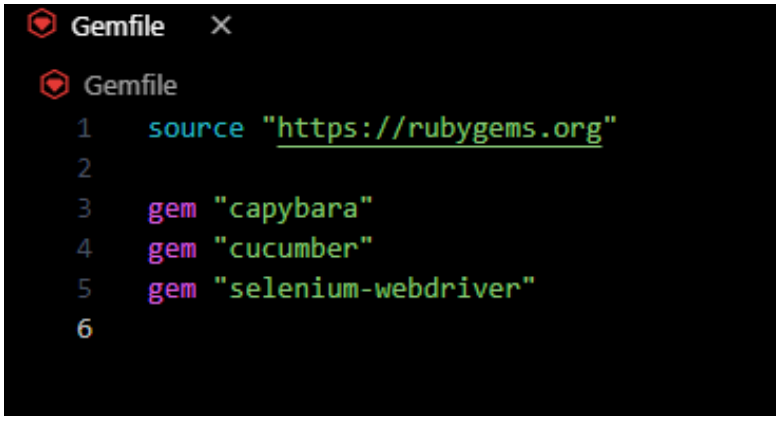
Fonte: Autor, 2019

A criação dos cenários por meio da linguagem Gherkin, permite descrever os cenários de testes em uma linguagem natural e de fácil entendimento para ambos desenvolvedores e analistas.

Com a criação dos cenários realizada, o *tester* irá realizar o comando “gem install bundler”, por meio do terminal, que irá instalar uma gem do Ruby que será responsável por gerenciar a instalação e atualização de novas gems, resolvendo de maneira facilitada as dependências de outras bibliotecas. Após isso, o *tester* deverá inicializá-lo através do comando “bundle init”, com isso será gerado o arquivo Gemfile que será usado para descrever as gems, suas versões, e de onde o bundler irá buscar pelas gems solicitadas pelo *tester*.

A figura a seguir apresenta o arquivo descrito:

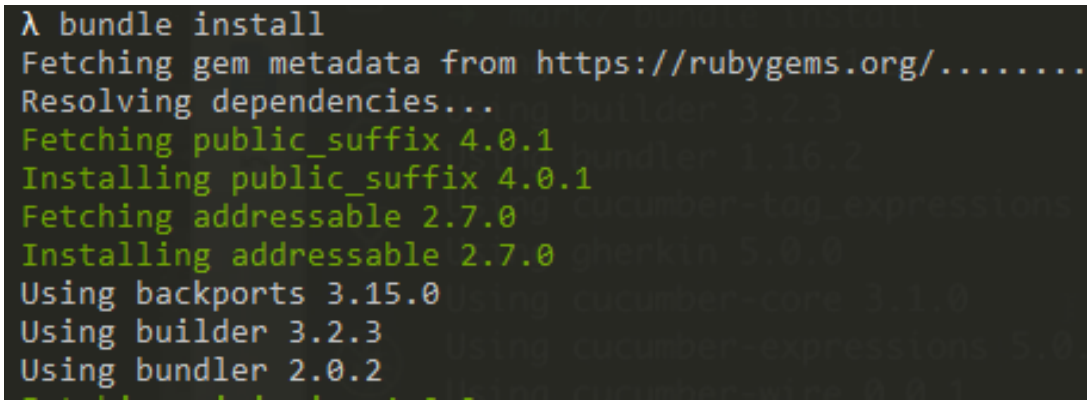
Figura 15 – Arquivo Gemfile



Fonte: Autor, 2019

Na figura 15 o source é a fonte onde o bundler irá buscar pelas gems, e cada gem possui seu papel específico na automação do *tester*, a gem "capybara" é baseada em selenium será responsável por validar certos testes em aplicações web, a gem "cucumber" é responsável pela utilização da ferramenta Cucumber pelo Ruby e a gem "selenium-webdriver" será responsável por abrir o navegador web. Para instalar as gems listadas, o *tester* executa o comando "bundle install", como descrito na figura a seguir.

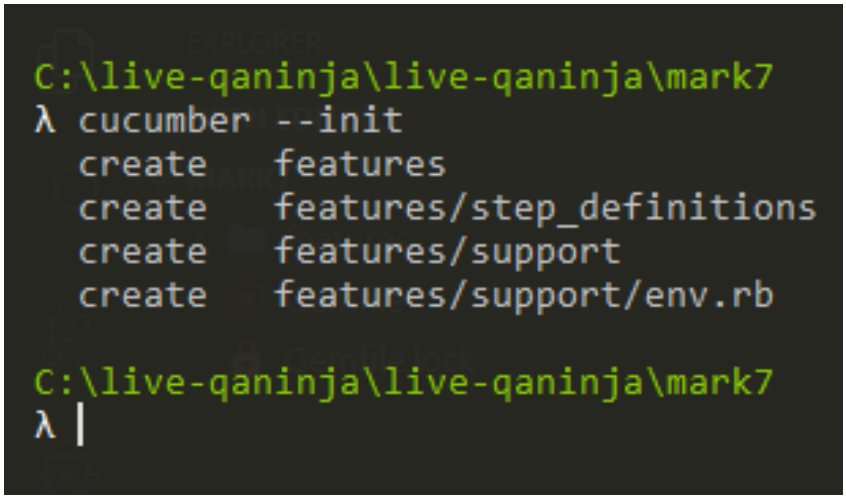
Figura 16 – Instalação de gems pelo bundle



Fonte: Autor, 2019

Após essa ação, todas as gems listadas no arquivo Gemfile foram instaladas com sucesso. Agora o *tester* deve executar a ferramenta cucumber, com o comando: "cucumber --init" que criará as seguintes pastas e arquivos:

Figura 17 – Inicialização do Cucumber



Fonte: Autor, 2019

O arquivo criado chamado de “env.rb” será responsável por executar tudo o que estiver descrito dentro dele em primeiro lugar, como por exemplo importar as gems instaladas no projeto, seus módulos e configurá-las.

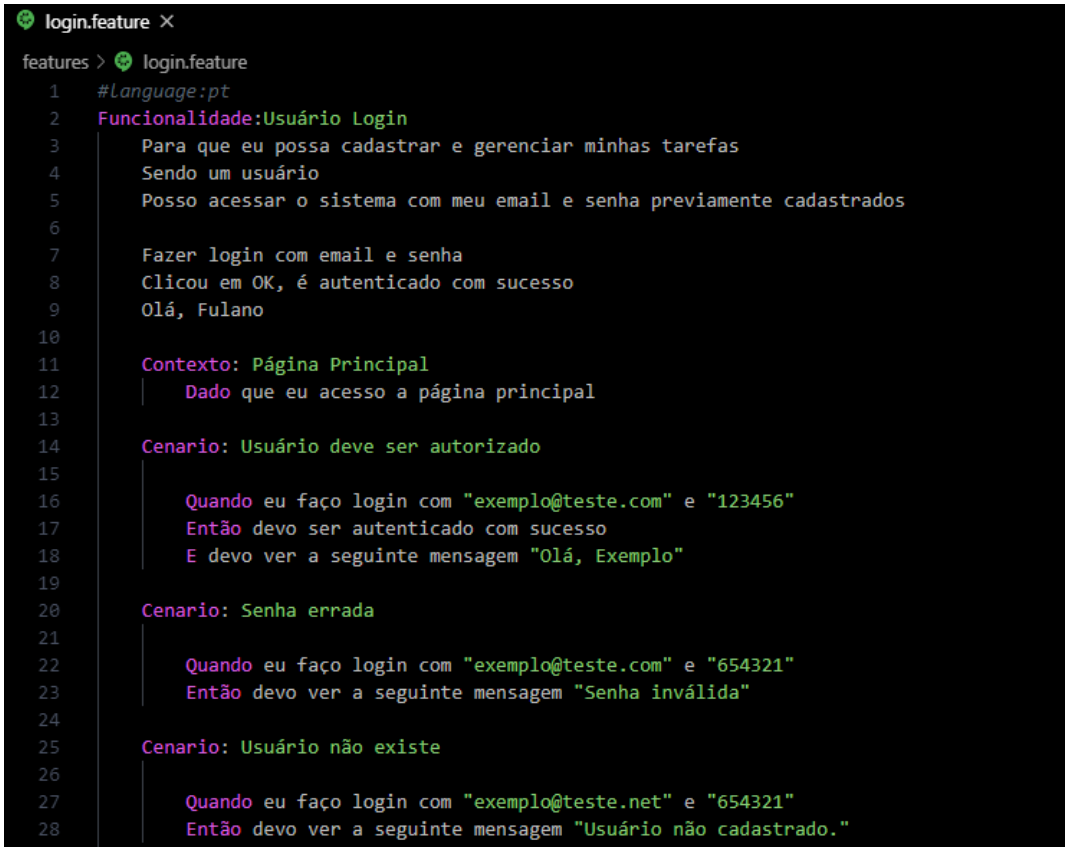
Figura 18 – Arquivo env.rb

Fonte: Autor, 2019

No arquivo em questão, é importada e configurada a gem do Capybara, seu módulo para trabalhar com o cucumber, e o navegador configurado para serem validados os testes é o Google Chrome. Além disso, é necessário o download do arquivo "chromedriver.exe" para que o navegador seja aberto corretamente, este arquivo mudará dependendo do navegador escolhido pelo *tester*.

Neste ponto o *tester* irá repassar a documentação com os cenários de testes, escritos com a sintaxe gherkin, para um novo arquivo com extensão "*.feature*" na pasta "*features*" como mostra a imagem:

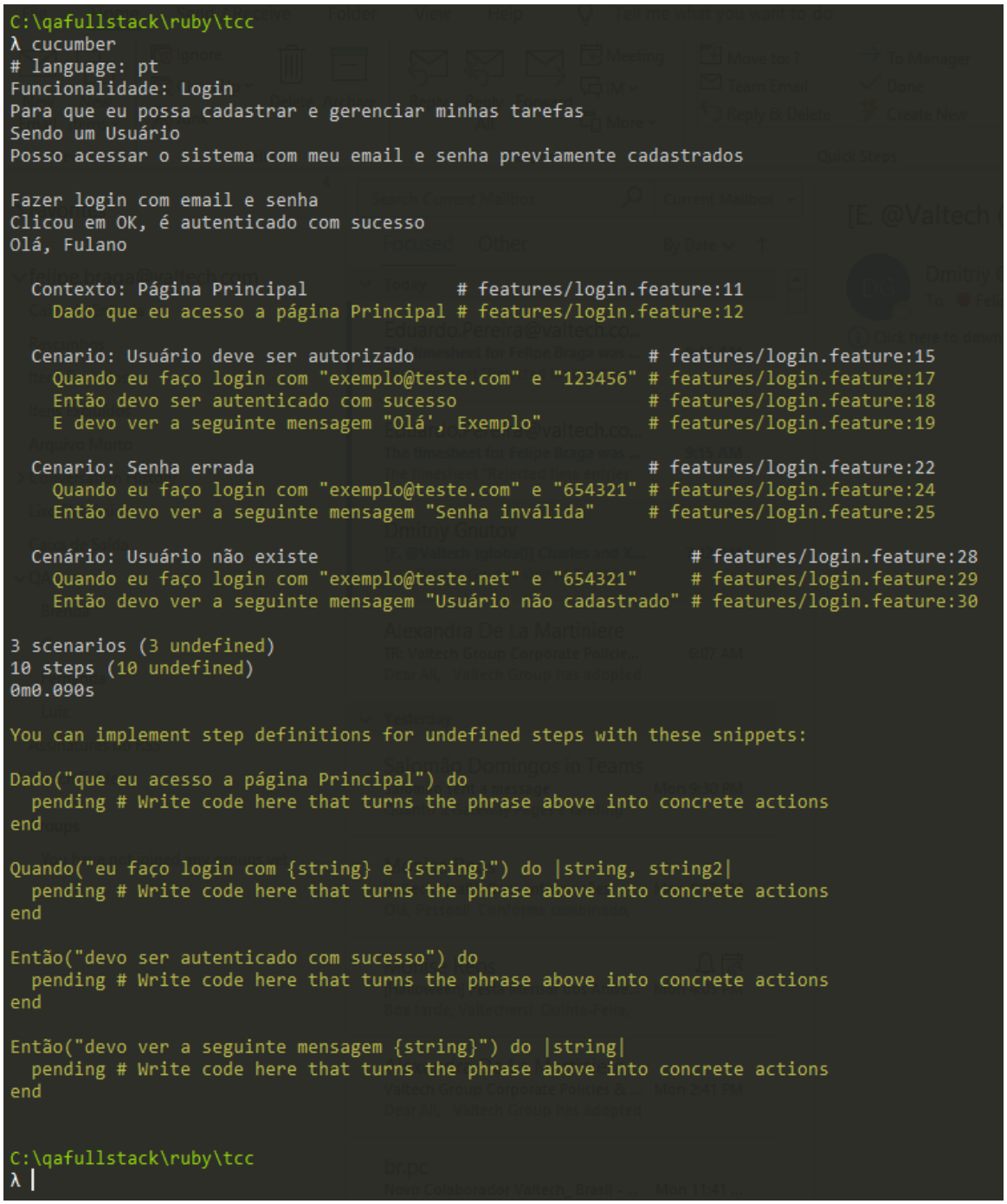
Figura 19 – Arquivo login.feature



Fonte: Autor, 2019

Após a criação do arquivo com os cenários de teste, é utilizada a ferramenta *Cucumber*, que possui a função de transcrever os cenários de testes em códigos funcionais através do comando "cucumber".

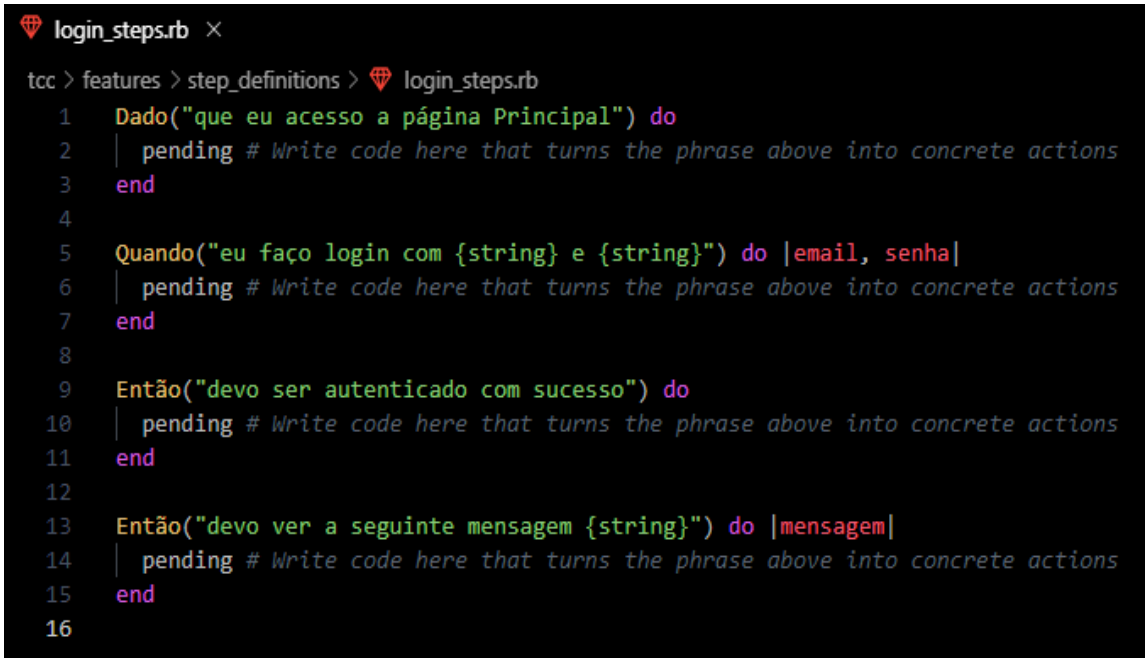
Figura 20 – Geração de códigos do Cucumber



Fonte: Autor, 2019

Os códigos gerados pelo Cucumber são mostrados no terminal para o *tester*, que os transfere para um novo arquivo chamado de “login_steps.rb” que será criado na pasta “step_definitions” mostrada anteriormente.

Figura 21 – Arquivo login_steps.rb



Fonte: Autor, 2019

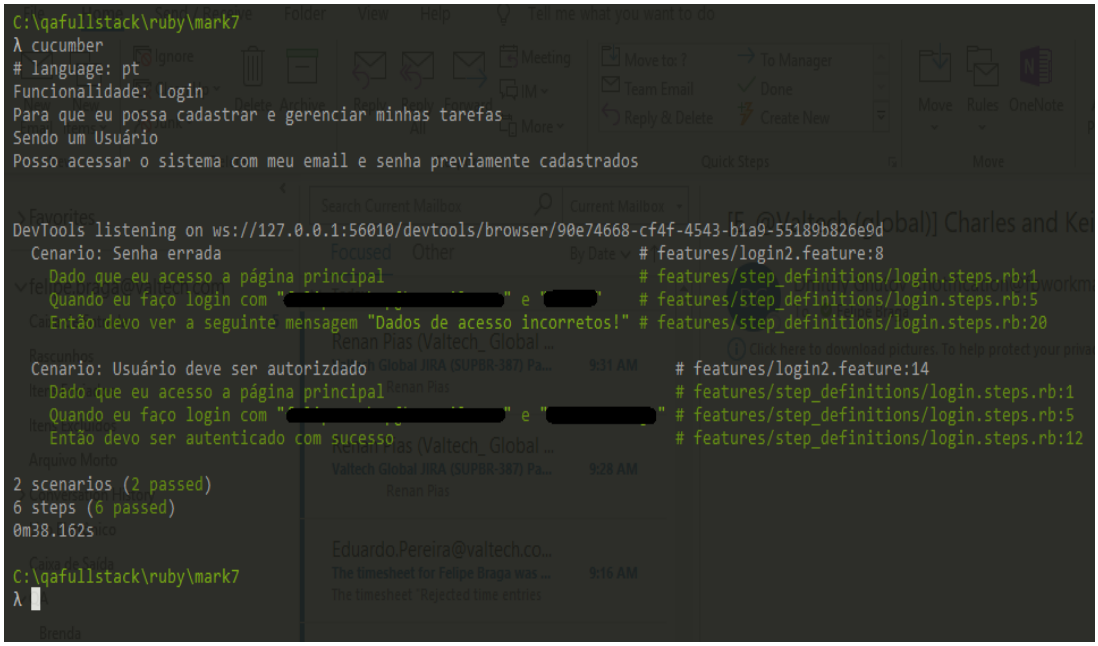
A partir deste momento caberá ao *tester* escrever os códigos de automação que quiser, respeitando os cenários e códigos gerados, sendo que cada linha de código representará um “step”, um passo traduzindo de maneira literal, da automação do teste.

Desta forma o *tester* é beneficiado por uma gama de possibilidades altíssimas devido a possibilidade de utilizar todos os métodos, códigos, comandos e validações das gems instaladas no “GemFile” do projeto.

Para ser executada a automação dos testes, como por exemplo visitar certa página, verificar se os elementos da documentação estão presentes na página, se as mensagens apresentadas pela página estão corretas, entre outros, codificadas pelo *tester*, é necessário abrir o terminal, chegar até a pasta do projeto e inserir o comando “cucumber”.

Com isso o cucumber irá realizar tudo o que foi descrito e/ou codificado no arquivo “login_steps.rb”, que utilizará as variáveis, por exemplo email e senha, com os valores declarados no arquivo “.feature”.

Figura 22 – Retorno de automação bem-sucedida

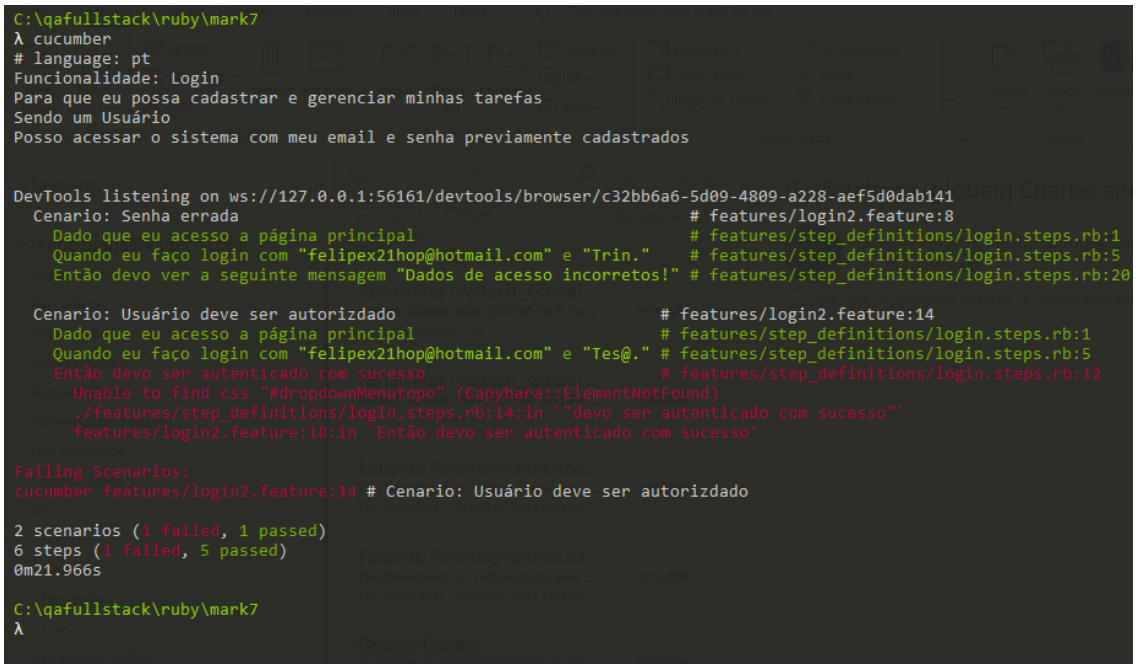


Fonte: Autor, 2019

O cucumber retorna os resultados dos cenários de teste para o *tester* informando se os testes foram concluídos com sucesso e passaram, ou se falharam e aponta em qual linha e em qual cenário ocorreu o erro.

Conforme mostra a imagem simulando um cenário que falhou ao ser executado:

Figura 23 – Retorno de automação com falha



Fonte: Autor, 2019

4.4.5 Considerações sobre a proposta

Devido ao projeto da empresa estudada já estar em andamento e com suas devidas prioridades de entrega do produto, a implantação da automação de testes não pode ser a prioridade da equipe no momento. A implantação da automação de testes requer tempo de preparação de ambiente, como por exemplo, junção de regras de negócios de todas as funcionalidades já desenvolvidas com o decorrer do projeto. Dessa forma é aconselhado que, logo após a reunião prévia do começo de *sprint*, o *tester* além de realizar a criação dos cenários de teste da *sprint* atual, também realize a automação dos cenários criados anteriormente ao decorrer do projeto.

Com a implantação da automação de testes, podem ser vistas melhorias consideráveis em questões de tempo e investimento nos testes, como por exemplo, testes de regressão, quando é necessário realizar testes em funcionalidades/aplicações testadas anteriormente para validar se as mesmas não foram impactadas. Com os testes automatizados será necessário automatizar apenas uma única vez e executar a automação quando for necessário a validação, e qualquer integrante da equipe poderá executar a automação e reportar os resultados gerados.

Porém a automação dos testes deverá ser focada nisso, tarefas repetitivas e que geram retrabalho para o *tester*. Quando é apresentado uma nova funcionalidade/aplicação, serão necessários testes de usabilidade, nesse caso a automação de testes não é apropriada, ela não garantirá que o sistema está funcionando completamente, por exemplo, se está visualmente agradável para o usuário. Além disso o *tester* também deve realizar cenários não previstos anteriormente, com o foco em analisar os comportamentos da aplicação através de fluxos alternativos, assim, possivelmente, encontrando um problema no software antes mesmo que este seja entregue para o cliente.

A implantação do novo processo de desenvolvimento será mais fácil que a implantação da automação de testes, devido ao modelo de desenvolvimento atual possuir certas características que podem ser ligadas ao Scrum, porém estão sendo aplicadas de maneira incorreta. Por exemplo, é estipulada uma data para a entrega do produto, porém não é visto com os desenvolvedores e *testers* o tempo que eles necessitarão para terminarem tais tarefas, assim sobrecarregando a carga horária de ambos para finalizarem e validarem o sistema até data marcada.

Com a implantação do novo processo de desenvolvimento, a equipe se aprimorará em inúmeros quesitos, como por exemplo, no quesito transparência, pois a equipe diariamente irá se comunicar contando o que cada um está fazendo no momento, se há algum impedimento, e o que foi realizado pelo colaborador no dia anterior, assim dando uma visão maior para toda equipe a respeito do andamento do projeto, se está atrasado ou adiantado.

Além disso, o time scrum estará motivado e empenhado para terminar as tarefas listadas para a *sprint*, pois as tarefas foram estimadas e acordadas na planning. Assim, cada tarefa concluída irá gerar o sentimento de progresso, o que não havia no antigo processo de desenvolvimento devido a novas inserções incessantes de tarefas gerando uma sensação de estagnação que desmotivava todos da equipe gerando atrasos e desconfiança entre desenvolvedores e analistas quanto ao comprometimento sobre o projeto.

Caso ocorra algum atraso no projeto, ou alguma funcionalidade não seja corretamente implantada, isso não será prolongado cada vez mais, a ponto de exercer uma cultura de achar culpados pelos erros. Através da *sprint retrospective*, na qual é avaliado por todo o time os pontos negativos e positivos da *sprint*, cada um pode apontar o que achou válido ou o que prejudicou o andamento da *sprint*, assim gerando uma melhoria contínua e constante da equipe a cada decorrer de *sprint*, para não serem cometidos os mesmos erros futuramente.

Dessa forma, neste capítulo foi apresentado a proposta do novo processo de desenvolvimento de software, com a descrição das atividades de cada membro do time, dando ênfase para as atividades de teste de software. Também foram detalhadas as ferramentas a serem usadas para a automação dos testes, aplicando as técnicas do BDD, e apresentando exemplos práticos.

5 CONCLUSÕES

Este trabalho teve como intuito apresentar o conceito de testes de software, sua importância no mercado atual, seus processos, estratégias e seus tipos, assim como, metodologias ágeis de desenvolvimento de software. O propósito era melhorar o processo da empresa estudada para realizar um entrega de produto com maior confiança e consistência para o cliente.

Para isso inicialmente foi realizada uma pesquisa bibliográfica sobre os seguintes assuntos: Engenharia de software, metodologias ágeis, Scrum, Kanban, qualidade de software, testes de software, seus processos, suas estratégias, tipos de teste, automação de testes, desenvolvimento orientado por comportamento, técnica do BDD, ferramentas de automação de testes para testes unitários, de integração, de interface e Selenium.

Essa teoria foi acompanhada pelo estudo e pesquisa de assuntos práticos e tecnológicos como por exemplo o estudo e aprendizado da linguagem Ruby. Ainda foi realizado o estudo e aprendizado da ferramenta *Cucumber* por meio da leitura de artigos, livros e cursos para a criação de cenários de teste utilizando a sintaxe Gherkin, e foi estudada a técnica BDD para incentivar o maior envolvimento da equipe de desenvolvimento e análise no processo de testes da aplicação.

No trabalho foi apresentada a descrição da empresa estudada, como as equipes são divididas, quantos colaboradores estão trabalhando no momento, seu processo de desenvolvimento atual, no qual é caracterizado o papel de cada membro da equipe.

A seguir foi apresentada a nova proposta de processo de desenvolvimento de software, baseada em um modelo ágil e no BDD, dando possibilidade para a automação de testes.

Foram listadas as principais ferramentas de automação de testes como Selenium, Appium, Cucumber, Robotium. Dentre essas foram escolhidas a ferramenta Cucumber e a linguagem Ruby para ser desenvolvida a automação no momento atual da empresa. Essa escolha foi devido a facilidade de aprendizado dessas ferramentas, pois existiam limitações de conhecimento por causa dos testes serem realizados somente de maneira manual e sem nenhuma codificação. Outros motivos da escolha foram: a) o tempo, não sendo necessário muitas horas para o estudo das tecnologias, logo não prejudicando o andamento do projeto, e b) investimentos, devido a serem ferramentas acessíveis. São listados como trabalhos futuros:

implementar a automação na empresa estudada, realizar a automação de testes em dispositivos moveis e realizar automação de testes de carga.

Finalmente foi apresentado que mesmo que a automação de testes traga diversos benefícios para a empresa, é constatado que ainda é necessário e recomendado serem realizados testes manuais e exploratórios.

REFERÊNCIAS

ANDRADE, Mayb. **Qualidade de software**. 1 ed. Rio de Janeiro: SESES, 2015.

BASTOS, A. et al. **Base de conhecimento em teste de software**. 2 ed. São Paulo: Martins Fontes, 2007.

BARTIÉ, Alexandre. **Garantia da qualidade de software: adquirindo maturidade organizacional**. Rio de Janeiro: Campus, 2002.

BAGGIO, Vinícius. **Ruby on Rails: Coloque Sua Aplicação Web nos Trilhos**. 1 ed. São Paulo: Casa do Código, 2012.

BERNARDO, Paulo Cheque; KON, Fábio. **A Importância dos Teste Automatizados**. 2008. Engenharia de Software Magazine, nº 3

CAROLI, Paulo. **Antologia Brasil: Histórias de Aprendizado e Inovação**. São Paulo: Casa do Código, 2014.

COSTA, Leandro Teodoro. **Conjunto de características para teste de desempenho: Uma visão a partir de ferramentas**. Porto Alegre, 2012.

CHAIM, Luis Fernando. **Cucumber – Linguagem Gherkin**. 2018. Disponível em <http://maxidica.com/wp/cucumber-linguagem-gherkin/> Acesso em: 06 out. 2019

DIOGO, Gabriel. **8 ferramentas de automação de testes que ajudam você na gestão de TI**. Disponível em <https://www.impacta.com.br/blog/2019/06/03/8-ferramentas-automacao-testes-ajudam-gestao-de-ti/> Acesso em: 08 set. 2019

FERREIRA, Avelino. **Como garantir a qualidade do software: Testes automatizados**. Disponível em <https://www.knowledge21.com.br/blog/qualidade-do-software-testes-automatizados> Acesso em: 09 set. 2019.

GOMES, Nelma da Silva. **QUALIDADE DE SOFTWARE – UMA NECESSIDADE**. Disponível em <http://www.unisalesiano.edu.br/salaEstudo/materiais/pd6952/material1.pdf> Acesso em: 16 set. 2018.

LUÍS, Bruno. **Melhoria de Conhecimentos em Garantia de Qualidade no Software (Tipos de Teste)**. Disponível em < https://slidex.tips/download/melhoria-de-conhecimentos-em-garantia-de-qualidade-no-software-tipos-de-teste>
Acesso em: 10 out. 2018

MARIOTTI, Flavio. **Kanban: o ágil adaptativo**. 2012. Engenharia de *Software* Magazine, nº 45

MARTINELLI, Fernando. **Gestão de qualidade total**. 2009.

PRESSMAN, Roger S. **Engenharia de Software: uma abordagem profissional**. 7. ed. São Paulo: Pearson Makron Books, 2011.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum**. 2013. Disponível em: <https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf> Acesso em: 08 out. 2018.

SILVA, E.L. da; MENEZES, E. M. **Metodologia da Pesquisa e elaboração de Dissertação**. 4 ed. UFSC, Florianópolis, 2005.

SOARES, Ismael. **Desenvolvimento orientado por comportamento (BDD)**. Disponível em: <https://www.devmedia.com.br/desenvolvimento-orientado-por-comportamento-bdd/21127> Acesso em: 15 set. 2018.

SOMMERVILLE, Ian. **Engenharia de software**. 9ª edição. São Paulo: Pearson Prentice Hall, 2011.

SOUZA, Lucas. **Ruby: Aprenda a programar na linguagem mais divertida**. São Paulo: Casa do Código, 2014.

WYNNE, Mat; HELLESOY, Aslak. **The Cucumber Book: Behaviour-Driven Development for Testers and Developers**. 2ª edição EUA: Jacquelyn Carter, 2017.