UNIVERSIDADE DO SUL DE SANTA CATARINA

MATHEUS FERNANDES

SAMUEL TOMKELSKI FONSECA

AUTOMAÇÃO DE TESTES DE SOFTWARE: ESTUDO DE CASO DA EMPRESA SOFTPLAN

Florianópolis

2019

MATHEUS FERNANDES
SAMUEL TOMKELSKI FONSECA

AUTOMAÇÃO DE TESTES DE SOFTWARE: ESTUDO DE CASO DA EMPRESA SOFTPLAN

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

Orientador: Maria Ines Castiñeira, Dra

Florianópolis, 2020

MATHEUS FERNANDES

SAMUEL TOMKELSKI FONSECA

AUTOMAÇÃO DE TESTES DE SOFTWARE: ESTUDO DE CASO DA EMPRESA SOFTPLAN

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Bacharel em Sistema de Informação e aprovado em sua forma final pelo Curso de Graduação em Sistema de Informação da Universidade do Sul de Santa Catarina.

Florianópolis, 06/2020

_____

Professor e orientador Maria Inés, Dra.
Universidade do Sul de Santa Catarina

_____

Prof. Richard Henrique de Souza, Dr.
Universidade do Sul de Santa Catarina

_____

Prof. Rafael Orivaldo Lessa, Esp.
Universidade do Sul de Santa Catarina

A Deus, que se mostrou criador, que foi criativo. Seu fôlego de vida em mim me foi sustento e me deu coragem para questionar realidades e propor sempre um novo mundo de possibilidades.

AGRADECIMENTOS

Matheus Fernandes agradece a:

Agradeço primeiramente a Deus, que com sua benção, permitiu que este trabalho pudesse ser realizado.

Agradeço a minha mãe, guerreira, que criou a mim e meus irmãos sozinha, mulher batalhadora a qual tenho muito orgulho de ser filho.

Agradeço a minha esposa por ter me dado o suporte necessário durante todos esses anos juntos.

Agradeço aos meus familiares e amigos pelo apoio. Agradeço ao meu amigo Samuel, pela parceria durante o curso e durante a elaboração deste trabalho.

Samuel agradece a:

Deus primeiramente por propiciar a conclusão do trabalho.

A toda minha família pelo apoio, principalmente minha mãe por ser o grande pilar nos momentos mais difíceis entre conciliar toda a rotina agitada do dia a dia.

Ao meu amigo Matheus que não mediu esforços para conciliar nossos horários, visto que ambos trabalham para possibilitar a conclusão do trabalho dando o nosso melhor.

A todos os professores da Unisul pelo aprendizado e parceria que fizeram os dias de aulas serem produtivos e estimulantes.

A Maria Inês por todo o suporte, conhecimento, paciência e orientação que possibilitou a direção do trabalho.

A Vera Schuhmacher por todo o apoio durante a graduação.

Aos convidados da banca que aceitaram o convite e dispuseram seu tempo para colaborar com o trabalho.

“Cada vez que a tecnologia avança o ser humano deve avançar na sua humanização, seja mais humano.” (RONALD PARADA).

RESUMO

A busca por maior produtividade com eficiência e qualidade é meta de toda empresa que tem o crescimento como um objetivo. Em uma empresa de *software* essa busca vai além, pois alcançar mais eficiência passa pela investigação e implementação das evoluções, quase diária, de novas tecnologias que possibilitam melhorar a qualidade de seus produtos. A automação veio para auxiliar o setor de testes dessas empresas a crescer justamente em produtividade e eficiência, sem de forma alguma renunciar à qualidade. Primeiramente o trabalho apresenta uma revisão da literatura falando sobre os processos de testes, tipos de testes, fase dos testes, plano de testes, automação de testes como também sobre ferramentas de automação mais utilizados. Com isso em mente, o presente trabalho tem o propósito apresentar como foi realizado o processo de automação de testes na equipe responsável pelo sistema solarBPM da empresa Softplan, elencando as principais atividades afetadas. A Softplan é uma das maiores empresas de software do Brasil, e o estudo foi desenvolvido dentro de uma equipe de testes de uma de suas várias áreas de atuação. O modelo de processo de desenvolvimento desse projeto é o modelo cascata, derivando as seguintes etapas: Elaboração de solução preliminar, elaboração de documento de requisitos, desenvolvimento, testes, homologação do cliente e finalmente produção. Nada obstante, neste cenário e na etapa de testes iniciou-se um projeto piloto de automação de testes. Além disso, para viabilização da automação de testes, fora comentado sobre a realização de um estudo das ferramentas disponíveis no mercado, bem como a elaboração de POC's com alguns frameworks como RobotFramework, Cucumber e Ranorex afim de avaliar a compatibilidade com as necessidades da equipe. Ademais, foram abordados os passos posteriores a POC, bem como explicar o conjunto de frameworks utilizados, demonstrando suas características através de seus conceitos e exemplos. O trabalho também discorre sobre as estratégias da automatização dos cenários para automação dos testes, bem como análises de alguns dados apresentados como gráficos e tabelas onde é possível observar os ganhos ou perdas com automação. Para complementar também foi abordado um pouco sobre as tecnologias que foram utilizadas no projeto e quais foram suas aplicações através de exemplos. Por fim, esse estudo ainda faz uma breve análise da automação realizada comparando-a com literatura estudada aliada com uma lista de recomendações de sugestões de melhorias que a equipe julga ser necessário para incorporar o que já foi realizado até o

momento. Isto posto, conclui-se na parte final do trabalho que a automação exige um investimento inicial tanto de recursos humanos quanto financeiros, mas possui um alto retorno de produtividade e agilidade na execução dos testes de médio a longo prazo. Com isso, foram sugeridas recomendações para aprimorar a automação de testes, bem como algumas propostas para trabalhos futuros.

Palavras-chave: Automação de Testes. Teste de Software. Estudo de caso. Softplan.

LISTA DE ILUSTRAÇÕES

Figura 1 - Ciclo do TDD 30
Figura 2 - Descoberta, Formulação e Automação 32
Figura 3 - Exemplo escrita BDD 33
Figura 4 - Processo de desenvolvimento 41
Figura 5 - Processo de teste 44
Figura 6- Test case - robot framework 47
Figura 7 - Interface ranorex 48
Figura 8 - Estrutura cucumber 50
Figura 9 - Exemplo sintaxe linguagem Ruby 51
Figura 10 - Implementação das pages em ruby 52
Figura 11 - Cenários transformados em linguagem de programação 53
Figura 12 - Cenários em BDD 54
Figura 13 - Exemplo de status report 1 - cucumber 55
Figura 14 - Exemplo de status report 2 - cucumber 55
Figura 15 - Exemplo de status report 3 - cucumber 56
Figura 16 - Comandos capybara 57
Figura 17 - Arquivar um processo administrativo 58
Figura 18 - Execução de teste - Git 60

LISTA DE GRÁFICOS

Gráfico 1 - Comparativo entre testes manuais x automação 62

LISTA DE QUADROS

Quadro 1 - Tipos de teste 24
Quadro 2 - Ferramentas de Automação mais utilizadas 28
Quadro 3 - Teste de regressão manual em horas 60
Quadro 4 - Teste de regressão automatizado 61

LISTA DE ABREVIAÇÕES

APF – Análise Ponto por Função.
BDD – Behavior Driven Development (Desenvolvimento Orientado a Comportamento).
DBA – Database administrator (Profissional Administrador do Banco de Dados).
GUI – Graphical user interface (Interface gráfica do usuário).
ISTQB – International Software Testing Qualification Board.
OTAN – Organização do Tratado do Atlântico Norte.
POC – Proof Of Concept - (Prova de Coiceito).
RUP – Rational Unified Process
SUT – System Under Test (Software em teste).

SUMÁRIO

1 INTRODUÇÃO 16
1.1 PROBLEMÁTICA 17
1.2 OBJETIVOS 18
1.2.1 Objetivo geral 18
1.2.2 Objetivos específicos 19
1.3 JUSTIFICATIVA 19
1.4 ESTRUTURA DA MONOGRAFIA 20
2 FUNDAMENTAÇÃO TEÓRICA 21
2.1 ENGENHARIA E QUALIDADE DE SOFTWARE 21
2.2 TESTES DE SOFTWARE 22
2.2.1 Processo de teste 23
2.2.2 Tipos de teste 23
2.2.3 Fases do teste 25
2.2.3.1 Testes Unitários 25
2.2.3.2 Testes de integração 26
2.2.3.3 Testes de sistema 26
2.2.3.4 Testes de Aceite 26
2.2.4 Plano de teste 27
2.3 AUTOMAÇÃO DE TESTE 27
2.3.1 Ferramentas e frameworks de automação 28
2.4 TÉCNICAS DE DESENVOLVIMENTO ÁGIL 29
2.4.1 TDD 29
2.4.2 BDD 31
3 METODOLOGIA DO TRABALHO 34
3.1 TIPO DE PESQUISA 34
3.2 ETAPAS DO DESENVOLVIMENTO DA PESQUISA 35
3.3 DELIMITAÇÕES DO TRABALHO 36
4 DESENVOLVIMENTO 37
4.1 DESCRIÇÃO DA EMPRESA 37
4.1.1 Justiça 37
4.1.2 Administração pública 38
4.1.3 Indústria da construção 38

4.1.4 Saúde 39
4.2 CARACTERIZAÇÃO DO PROJETO 39
4.3 PROCESSO DE DESENVOLVIMENTO E TESTES 39
4.3.1 Processo de Desenvolvimento 40
4.3.1.1 Elaboração da solução preliminar 42
4.3.1.1.1 Análise de Pontos por função 42
4.3.1.2 Elaboração do documento de requisitos 43
4.3.1.3 Desenvolvimento 43
4.3.1.4 Testes 43
4.3.1.5 Homologação do cliente 44
4.3.2 Processo de Testes 44
4.3.2.1 Estimativa de testes 45
4.3.2.2 Especificação de testes 45
4.3.2.3 Execução de testes 46
4.4 PROCESSO DE AUTOMAÇÃO 46
4.4.1 Estudo das ferramentas 46
4.4.2 Elaboração das POCs 47
4.4.3 Apresentação para a equipe 49
4.4.4 Automação 49
4.5 DESCRIÇÃO E ANÁLISE DA IMPLEMENTAÇÃO DA AUTOMAÇÃO 57
4.6 ANÁLISE DA AUTOMAÇÃO À LUZ DA TEORIA ESTUDADA 63
4.6.1 Analisando ponto a ponto 63
4.6.2 Lista de Recomendações e Melhorias 64
5 CONCLUSÕES 66
REFERÊNCIAS 68

1 INTRODUÇÃO

Em tempos em que a tecnologia avança de maneira extraordinária, cada vez mais novos sistemas, softwares, hardwares, aplicativos e afins são produzidos. Praticamente todos os anos uma novidade tecnológica é lançada. Como exemplo podemos citar o setor da tecnologia focado no âmbito de *smartphones.* Só no ano de 2019 Apple fez o lançamento de três *smartphones* diferentes, entre eles o iPhone 11, que está com as pré-vendas no site oficial da Apple (CAPELAS, ROMANI, WOLF, 2019).

A partir da dessas informações, pode-se questionar como esses sistemas, aplicativos, são de fato testados. E mais, quantos desses sistemas possuem algum tipo de automação de testes referentes ao processo de desenvolvimento.

O autor de Engenharia de Software, Pressman (2016, p. 412) evidenciou a necessidade de maior qualidade de software, conforme a tecnologia e os sistemas informatizados passaram a fazer parte de nossas vidas cotidianas. Além de ressaltar que na década de 50, bilhões de dólares eram desperdiçados com softwares que não atendiam aos comportamentos esperados pelos clientes.

Para a Standish Group – empresa de pesquisa de mercado (2016, p. 412 apud) "O código malfeito continua a ser o "fantasma" do mercado de software, sendo responsável por até 45% do tempo de inatividade dos sistemas computacionais e custando as empresas americanas cerca de U$ 100 bilhões no último ano. Isso não inclui o custo da perda de clientes insatisfeitos." A respeito desse mesmo assunto, Maheshwari (2012, p. 285) corrobora afirmando que todos os sistemas de software são imperfeitos porque não podem ser construídos com recursos matemáticos ou físicos.

Para evitar perda tanto de tempo, quanto de dinheiro, as empresas, com o tempo, foram observando a necessidade de realizar testes de software em suas produções antes de definir como completo um sistema. Com essa observação, a área de testes começou a crescer e hoje faz parte de todo o processo de desenvolvimento de um software.

Sommerville (2007) descreve um reflexo triste na época de sua publicação, quando muitos usuários tinham expectativas baixas e não se surpreendiam com as falhas no uso do sistema. Segundo ele, os usuários tendiam a aceitar as falhas dos sistemas quando as vantagens superavam as dificuldades do uso. Ainda assim, menciona que já desde a década de 1990 a tolerância a falhas vem diminuindo, sendo menos aceitável entregar sistemas não confiáveis. Raim (2010) também reforça essa mudança comportamental destacando a independência das

pessoas que utilizam softwares, desde um usuário doméstico a usuários corporativos que dependem 100% desses sistemas.

O ISTQB, International Software Testing Qualifications Board (2018), aponta que quando são feitas alterações em um sistema, seja para corrigir um defeito ou por causa de uma funcionalidade nova ou variável, deve-se testar para confirmar se as alterações corrigiram o defeito ou implementaram a funcionalidade corretamente e não causaram consequências adversas imprevistas.

A consultora de sistemas (SOFTDESIGN, 2017) esboça que os Softwares em desenvolvimento requerem um grande esforço em testes. Quando se é executado um mesmo cenário de teste inúmeras vezes, isso pode ocasionar erros, como por exemplo, pular algum passo importante do cenário. Araújo (2018) reforça que é possível fazer todos esses testes manualmente, mas isso é moroso e tedioso, ou seja, testes manuais podem ser maçantes, demandando muito tempo, gerando custos, já que são repetidos inúmeras vezes, quando for necessário. O autor transcreve que somos humanos e estamos suscetíveis a errar. Para auxiliar nesse contexto, pode-se implementar automações de testes de software. A automação vem para fornecer uma alternativa confiável, rápida e eficaz, mas necessita de um esforço inicial para alcançá-la, da qual não será atingida com facilidade, ou até mesmo não será atingida apenas com testes manuais (BARTIÉ, 2002).

O ISTQB (2018) indica, portanto, que os conjuntos de testes de regressão que são executados muitas vezes, e geralmente evoluem lentamente, poderão ser um forte candidato à automação.

1.1 PROBLEMÁTICA

Oliveira (2018) discorre sobre o papel da automação de testes que objetiva a redução do envolvimento humano, tempo e custo final. A automatização então busca preencher essas lacunas relacionadas, por exemplo, com os testes de regressão manuais. Além disso, a autora também ressalva que “ao pensar em automatizar, é preciso estudar a viabilidade”, ou seja, é necessária uma análise das vantagens e desvantagens de automatizar um software, cujo principais pontos a serem atendidos são:

1. Há ganho de tempo?
2. É possível reduzir os custos a longo prazo?
3. Manterá a qualidade?

Atendendo positivamente as questões supracitadas, Oliveira (2018) atenta que “outros fatores precisam ser analisados: a maturidade do time de processo do teste; grau de reutilização dos testes automatizados; conhecimento sobre o comportamento que é esperado do sistema a ser testado; e, ainda, o tempo disponível para a automação”, ou seja, não é uma tarefa fácil, devendo ter um trabalho de análise crítico do nível de maturidade do time, nível de capacitação das pessoas em operar o software.

Ao respeito, Caetano (2018) complementa que “A automação de testes é um investimento com retorno garantido quando aplicada corretamente. No entanto, o retorno é de longo prazo, ou seja, não existe retorno imediato.” Dessa forma, o tempo de investimento em automação é de médio a longo prazo. Assim a problemática apontada por este trabalho se mostra pela dificuldade que algumas empresas têm em entender que a automação de testes é necessária quando existe uma repetição de trabalho, ou seja, pontos onde seria mais fácil automatizar, porém é deixado de lado por causa do investimento inicial necessário.

1.2 OBJETIVOS

A seguir, os autores do trabalho descrevem quais são seus objetivos com o desenvolvimento deste trabalho:

1.2.1 Objetivo geral

Apresentar um estudo de caso sobre automatização de testes de software na empresa de desenvolvimento Softplan.

1.2.2 Objetivos específicos

- Realizar uma pesquisa bibliográfica sobre qualidade, testes de software e automatização desse processo.
- Estudar como a automação de testes pode auxiliar à equipe quanto redução de tempo e cobertura de testes.
- Descrever os processos de desenvolvimento e de testes, assim como as tecnologias e ferramentas utilizadas no produto do estudo de caso da empresa Softplan.
- Destacar a importância e os desafios na automatização de testes de software.
- Comparar os testes manuais e automatizados apresentando as suas vantagens e desvantagens.
- Analisar à luz da literatura os ganhos obtidos e as melhorias que ainda podem ser realizadas no processo estudado, gerando uma série de recomendações.

1.3 JUSTIFICATIVA

De acordo com Pressman (2016) “A necessidade de maior qualidade de software surgiu realmente a partir do momento que o software ficou cada vez mais integrado a todas as atividades de nossas vidas.” Atualmente os usuários estão cada vez mais conectados e interagindo com softwares de diversos segmentos que fazem parte de seu dia a dia, principalmente os jovens que já nascem como nativos digitais. Abreu (2019) complementa dizendo que “Nas últimas décadas, o usuário se tornou cada vez mais exigente. Se antes ele tinha poucas opções de marcas e pouco conhecimento sobre produtos e tecnologias, hoje ele tem toda informação na palma da mão”.

Em paralelo surgem novas tecnologias e conceitos pensados no usuário que aprimoram a relação entre o humano e um software, conforme ressaltado por Abreu (2019) ao afirmar que "Assim como outros departamentos, a área de TI também deixou de olhar apenas para softwares e infraestrutura e passou a focar na qualidade da experiência do usuário". Dessa forma, as pessoas que entram em contato com essas técnicas possuem baixa tolerância a falhas, ficando mais exigentes.

Conforme ISTQB (2018, p. 13):

> A maioria das pessoas já teve uma experiência com software que não funcionou como o esperado. O software que não funciona corretamente pode levar a muitos problemas, incluindo a perda de dinheiro, de tempo ou da reputação comercial, e até mesmo ferimentos ou morte. O teste de software é uma maneira de avaliar a qualidade do software e reduzir o risco de falha do software em operação.

Sendo assim, é necessário ressaltar a importância do valor da qualidade de software que visa atender a expectativa dos usuários, podendo mitigar constrangimentos e os prejuízos caso não seja dado a devida atenção.

1.4 ESTRUTURA DA MONOGRAFIA

No capítulo 1, o trabalho tem início com uma introdução a respeito do tema abordado pela equipe, testes automatizados. A justificativa e os objetivos deste trabalho também se encontram nesta primeira parte.

Já no segundo capítulo, o trabalho aborda toda a parte de fundamentação teórica descrevendo os conceitos e conteúdos necessários para o procedimento desta monografia.

No capítulo 3 é apresentado o método do trabalho, com a conceituação do tipo de pesquisa, etapas metodológicas e delimitações do trabalho.

No quarto capítulo, os desenvolvedores deste trabalho dão início ao desenvolvimento com a descrição do estudo de caso.

Finalmente, o capítulo 5 apresenta todas as conclusões que os autores deste trabalho chegaram, conforme os resultados obtidos através do estudo de caso.

2 FUNDAMENTAÇÃO TEÓRICA

Será abordado neste capítulo, toda a teoria usada como base e fundamentação para o desenvolvimento deste trabalho.

2.1 ENGENHARIA E QUALIDADE DE SOFTWARE

Para poder entender este trabalho como um todo, é necessário ter em mente o entendimento de alguns conceitos básicos, entre eles, o software. Para trazer um enfoque para esse e outros conceitos, pode-se começar entendendo o que é a engenharia de software e o que ela representa. Podemos citar a definição proposta por Fritz Bauer ainda na década de sessenta. “O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais” (NAU69 apud PRESSMAN, 1995, p.31).

Pressman argumenta ainda que apesar de existirem definições mais abrangentes, todas elas concordam na importância da disciplina de engenharia no processo de construção de um software. Sommerville (2011) discorre que a engenharia de software é uma disciplina da engenharia onde estão todos os aspectos do desenvolvimento de um sistema, apontando desde os fundamentos básicos na definição de um sistema, até pontos que abrangem a manutenção.

A engenharia de software foi formalizada na conferência de Engenharia de Software, no ano de 1968, pelo comitê de ciências da OTAN (Organização do Tratado do Atlântico Norte). Essa conferência teve como objetivo discutir o a crise do software que existia na época, essa crise fazia referência a dificuldade de se escrever programas com um padrão e que pudessem ser revistos e verificados. Nessa conferência foram discutidos três pontos básicos da engenharia de software.

- Processos mais previsíveis.
- Processos mais produtivos.
- Melhor qualidade.

A respeito da qualidade que o software deve possuir, Pressman sua definição sobre o tema, dizendo que a qualidade de software “é a conformidade com requisitos funcionais e de desempenho explicitamente declarados, padrões de desenvolvimento explicitamente documentados e características implícitas, que são esperadas em todo software desenvolvido profissionalmente" (PRESSMAN, 2002).

Alexandre Bartié (2002) também define o que seria a qualidade de software quando diz que é um processo sistemático que tem o seu foco em todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos.

Dentro da qualidade de software, existem diversas formas de se garantir que ela será alcançada, uma delas é através dos testes de softwares.

2.2 TESTES DE SOFTWARE

Para tentar garantir que um software funciona exatamente da forma em que foi planejado, existem os alguns meios, um deles é o teste. A fase de testes faz parte do desenvolvimento de um software, e ele é uma das formas que as empresas podem garantir a qualidade do sistema. Por isso, quando falamos em testes, falamos em garantia de qualidade.

2.2.1 Processo de teste

Processo de teste pode ser definido como “um conjunto de passos parcialmente ordenados, cujo objetivo é atingir uma meta para entregar um produto de software de maneira eficiente previsível e que vá ao encontro das necessidades do negócio” (MOLINARI, 2008 pág. 74). Essas necessidades de negócio são definidas nos processos de testes de acordo com diversas variáveis de um contexto, pois conforme ISTQB (2018) não existe um processo de testes consolidado e universal, mas há um conjunto comum de atividades de testes para aumentar a probabilidade de atingir os limites estabelecidos.

Para ISTQB (2018, p.18) os principais fatores que contextuais que influenciam nos processos de testes são:

- Modelo de ciclo de vida de desenvolvimento de software e metodologias de projeto utilizados.
- Níveis de teste e tipos de teste considerados.
- Riscos de produto e projeto.
- Domínio do negócio.
- Algumas restrições operacionais:
  - Orçamento e recursos.
  - Escalas de tempo.
  - Complexidade.
  - Requisitos contratuais e regulamentares.
  - Políticas e práticas organizacionais.
- Normas internas e externas necessárias.

2.2.2 Tipos de teste

Existem vários tipos de testes de software, cada um com uma finalidade definida. ISTQB (2018) define que um tipo de teste é um grupo de atividades de teste destinado a testar

características específicas de um sistema de software, ou parte de um sistema, com base em objetivos de teste específicos.

Quadro 1 - Tipos de teste

Tipo de Teste	Descrição
Teste de unidade	Teste em um nível de componente ou classe. É o teste cujo objetivo é um "pedaço do código".
Teste de integração	Garante que um ou mais componentes combinados (ou unidades) funcionam. Podemos dizer que um teste de integração é composto por diversos testes de unidade.
Teste operacional	Garante que a aplicação pode rodar muito tempo sem falhar.
Teste positivo-negativo	Garante que a aplicação vai funcionar no "caminho feliz" de sua execução e vai funcionar no seu fluxo de exceção.
Teste de regressão	Toda vez que algo for mudado, deve ser testada toda a aplicação novamente.
Teste de caixa preta	Testar todas as entradas e saídas desejadas. Não se está preocupado com o código, cada saída indesejada é vista como um erro.
Teste de caixa branca	O objetivo é testar o código. Às vezes, existem partes do código que nunca foram testadas.
Teste funcional	Testar as funcionalidades, requerimentos, regras de negócio presentes na documentação. Validar as funcionalidades descritas na documentação (pode acontecer de a documentação estar inválida)
Teste de interface	Verifica se a navegabilidade e os objetivos da tela funcionam como especificados e se atendem da melhor forma ao usuário.
Teste de performance	Verifica se o tempo de resposta é o desejado para o momento de utilização da aplicação.
Teste de carga	Verifica o funcionamento da aplicação com a utilização de uma quantidade grande de usuários simultâneos.
Teste de aceitação do usuário	Testa se a solução será bem vista pelo usuário. Ex: caso exista um botão pequeno demais para executar uma função, isso deve ser criticado em fase de testes. (aqui, cabem quesitos fora da interface, também).
Teste de volume	Testar a quantidade de dados envolvidos (pode ser pouca, normal, grande, ou além de grande).

Teste de stress	Testar a aplicação sem situações inesperadas. Testar caminhos, às vezes, antes não previstos no desenvolvimento/documentação.
Teste de configuração	Testar se a aplicação funciona corretamente em diferentes ambientes de hardware ou de software.
Teste de instalação	Testar se a instalação da aplicação ocorreu corretamente.
Teste de segurança	Testar a segurança da aplicação das mais diversas formas. Utilizar os diversos papéis, perfis, permissões, para navegar no sistema.

Fonte: Portal GSTI, 2019

Dentre os tipos de testes citados acima, existem alguns que são mais recomendados a serem automatizados do que outros. Os testes de regressão são um dos principais tipos de testes a serem automatizados, pois geralmente são repetitivos e de longa execução (ISTQB, 2018).

2.2.3 Fases do teste

São quatro as fases de testes: Unitários, integração, sistema e aceitação. Essas fases podem ser encontradas por outros termos como estágios ou níveis. O manual ISTQB define que as fases são um grupo de atividades de teste que são organizados e gerenciado juntos. Além disso, salienta que as fases estão relacionadas com outras atividades dentro do ciclo de vida de desenvolvimento de software.

2.2.3.1 Testes Unitários

Segundo ISTQB (2018) os testes unitários ou de componentes verificam o funcionamento de partes do sistema de forma isolado ou separada. Rios e Moreira (2013) afirmam que os testes unitários é a fase inicial da escala de testes aplicados nos menores componentes criados, visando garantir que estes atendam as especificações, em termos de características e funcionalidade.

2.2.3.2 Testes de integração

O teste de integração basicamente se concentra nas interações entre componentes ou sistemas. Quando o sistema se comunica com vários sistemas externos por meio de diferentes protocolos, ou se a aplicação em si consiste em uma série de módulos de baixo acoplamento com interações complexas entre eles, então testes de integração tornam-se muito importantes. Os testes de integração garantem que cada parte independente de sua aplicação funciona corretamente com os serviços dos quais depende (HUMBLE; FARLEY, 2014).

2.2.3.3 Testes de sistema

Rios e Moreira (2013) discorre que o teste de sistema tem como objetivo a execução do sistema como um todo ou um subsistema (parte do sistema), dentro de um ambiente operacional controlado, para validar a exatidão e perfeição na execução de suas funções. Além disso, expõe que devem ser testadas todas as funções cujo ambiente seja mais próximo ao de produção.

ISTQB (2018) corrobora ao explicar que o teste de sistema se concentra no comportamento e nas capacidades de todo um sistema ou produto, geralmente considerando as execuções das tarefas de ponta a ponta do sistema e os comportamentos não funcionais exibidos ao executar tais tarefas.

2.2.3.4 Testes de Aceite

Os testes de aceite se baseiam no comportamento ou na capacidade do sistema como um todo, isto é, verificar se os requisitos funcionais e não funcionais do sistema são os especificados, tal qual lembra Sommerville (2017) que os testes de aceitação é tipo um teste de usuário e reforçado por Pressman (2016) ao dizer que “o teste de aceite ou validação tenta descobrir erros, mas o foco está no nível de requisitos – em coisas que ficarão imediatamente

aparentes para o usuário, focalizando ações visíveis ao usuário e saídas do sistema reconhecíveis pelo usuário.

2.2.4 Plano de teste

O plano de teste serve para colocar em prática os pontos levantados no processo de testes. Cem Kaner (apud MOLINARI 2008, pág 93) conceitua de forma bem clara como o plano de teste é utilizado no dia-a-dia. “Plano de testes é um conjunto de ideias que guiam ou representam o processo de teste. Muitas dessas ideias são documentadas parcialmente, espalhadas em múltiplos documentos, estando sujeitas às mudanças conforme o projeto evoluir”.

O plano de teste sempre é desenvolvido com algum objetivo. Atualmente pode-se citar as seguintes principais funções do plano de teste:

- Dar suporte ao desenvolvimento.
- Descrever e justificar a estratégia de teste.
- Dar suporte ao início e a organização do projeto de testes.
- Identificar as entregas do projeto.
- Ter um histórico das informações do projeto para melhorias e auditorias.
- Suportar o gerenciamento diário juntamente com a evolução do projeto.

2.3 AUTOMAÇÃO DE TESTE

Molinari afirma que existem duas razões iniciais para a automatizar testes de software: O primeiro argumento tem relação ao crescimento elevado de sistemas e aplicações, onde se torna difícil o controle e elaboração de testes manuais. Já a segunda razão, baseia-se na alta complexidade dos softwares que são desenvolvidos atualmente. Com isso em mente, e considerando os pontos levantados, existem sistemas em que a única solução para a execução dos testes é a utilização da automatização dos testes, levando em consideração o tipo de teste

que será realizado e da função a ser realizada, tornando assim algumas vezes a execução do teste manual inviável (MOLINARI, 2008).

O profissional que trabalha com automação de testes, precisa entender da programação do sistema desenvolvido e a lógica de implementação que existe por trás do sistema, sendo que anteriormente, essa necessidade pertencia aos programadores. Outra necessidade é o conhecimento das funcionalidades e características do sistema onde será utilizada a automatização, pois muitas vezes, possuem linguagens exclusivas para o seu desenvolvimento (MOLINARI, 2008).

2.3.1 Ferramentas e frameworks de automação

De acordo com o que diz o ISTQB (2018), as ferramentas de automação de testes funcionam com a execução de um conjunto de instruções redigidas em uma linguagem de programação, que na maioria das vezes é chamada de linguagem script. As instruções para as ferramentas são muito detalhadas e entradas específicas, ordem das entradas, valores específicos utilizados nas entradas e saídas esperadas. Isto pode deixar scripts detalhados suscetíveis a alterações no software em teste (SUT, na sigla em inglês), particularmente quando a ferramenta interage com a interface gráfica de usuário (GUI, na sigla em inglês).

No contexto do desenvolvimento de um sistema, um software ou uma aplicação, existem diversas ferramentas que podem ser utilizadas para realização da automação de testes. Entre elas estão o Selenium, Cucumber, Apium, TestComplete, Ranorex, Robotium, entre outras. Cada uma possui elementos e funcionalidades que as tornam específicas para determinados tipos de teste e ambientes. (CONTROL, 2018).

Quadro 2 - Ferramentas de Automação mais utilizadas

Ferramenta	Ambiente	Linguagem
SELENIUM	Web	Java, Perl, JavaScript, PHP, Python, C#, Ruby
CUCUMBER	Web	Ruby,Java, NET, Scala, Groovy
TESTCOMPLETE	Desktop, Mobile, Web	JavaScript, Python, JScript, Delphi, C++ e C#

TELERIK TS	Mobile, Web	Angular, Asp.Net, HTML5, JavaScript, Ajax, Silverlight, Ruby, PHP, VB.Net e C#.
ROBOTIUM	Mobile	Java
RANOREX	Desktop, Mobile, Web	Não utiliza liguagem
APIUM	Mobile	Java
TESTINGWHIZ	Web, Mobile, BDD	Não utiliza liguagem

Fonte: CONTROL, 2018

2.4 TÉCNICAS DE DESENVOLVIMENTO ÁGIL

Abaixo serão relatadas duas técnicas de desenvolvimento ágil. Como será visto, o BDD tem sua origem a partir do TDD. Portanto serão analisadas algumas características dessas duas técnicas.

2.4.1 TDD

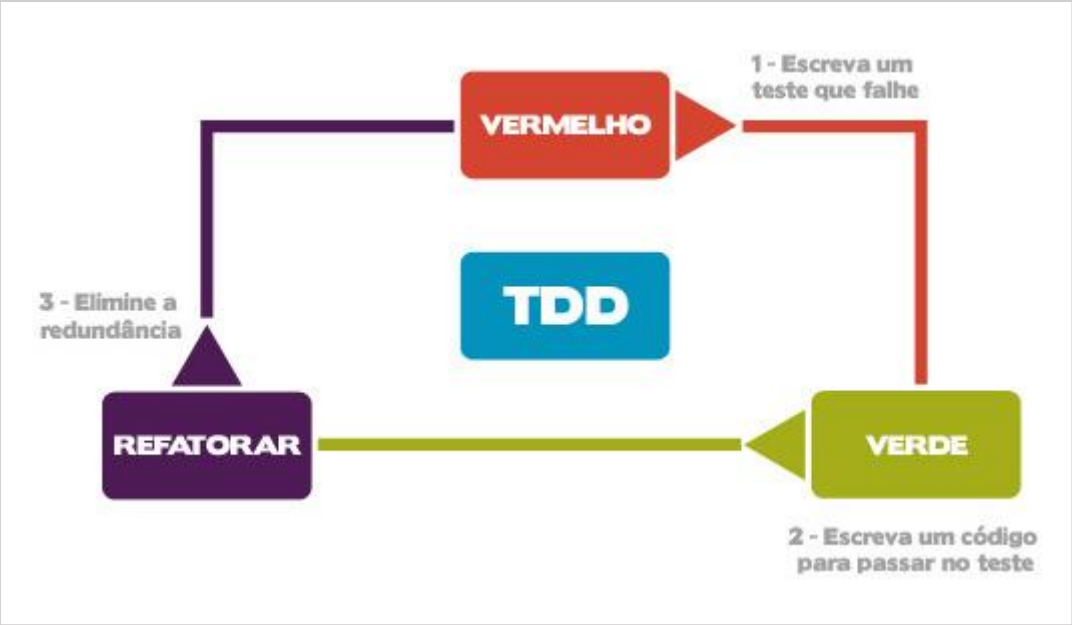
De acordo com Beck (2002) - autor da técnica de desenvolvimento dirigido por testes - o TDD (Test Drive Development) é uma técnica de desenvolvimento de software que se dá pela repetição de um ciclo curtos com passos de implementação de testes e do sistema" (apud BERNARDO, 2011, p. 155).

Ainda em conformidade ao que afirma Bernardo (2011) esse ciclo do TDD é dividido em três passos:

1. Implementar apenas um caso de teste;
2. Implementar apenas um trecho de código de produção que seja suficiente para o novo caso de teste ter sucesso de tal modo que não quebre os testes previamente escritos;

3. Se necessário, refatorar o código produzido para que ele fique mais organizado. Do mesmo modo, não deve interferir nos testes já implementados, além disso, essa refatoração não deve alterar as funcionalidades do sistema.

Figura 1 - Ciclo do TDD



Fonte: DEVMEDIA, 2013

Conforme pode-se verificar na figura acima, o ciclo do TDD é curto, porém a repetição desse ciclo é quem garante a funcionalidade dessa técnica de desenvolvimento. Em resumo o TDD é uma prática que ajuda a controlar e garantir a qualidade de um sistema. Ela sugere uma forma simples de desenvolvimento com o enfoque na prevenção de erros e na criação de sistemas bem modularizados. (BERNARDO, 2011 pág. 156).

Aniche (2014) reafirma os pontos principais e explana um pouco mais sobre como funciona o TDD na prática:

> [...] escreva um teste que falha, faça-o passar da maneira mais simples possível e, por fim, refatore o código [...] Sempre que um desenvolvedor pega uma funcionalidade para fazer, ele a quebra (muitas vezes mentalmente) em pequenas tarefas. Tarefas essas que exigem a escrita de código. Classes são criadas, outras são modificadas. Todas essas modificações tem um propósito, claro. Todo código escrito tem um objetivo. Ao praticar TDD, o desenvolvedor antes de começar a fazer essas modificações explicita esses objetivos. Só que ele faz isso por meio de testes

> automatizados. O teste em código nada mais é do que um trecho de código que deixa claro o que determinado trecho de código deve fazer.

2.4.2 BDD

Arcanjo (2015) relata que o BDD (Behavior Driven Development, que traduzido ao português significa “Desenvolvimento orientado por comportamento) foi inventado por Dan North no ano de 2000. Dan percebeu que muitas equipes tinham dificuldades de adotar e usar eficazmente o TDD, criado como uma versão melhorada do desenvolvimento orientado por testes (TDD, criado por Kent Beck).

Além disso, (Arcanjo, 2015) também discorre sobre o BDD não ser uma metodologia de desenvolvimento de software, como tão pouco um substituto para XP, Scrum, Kamban, *OpenUP*, RUP ou qualquer outra metodologia que o mercado oferece. O autor argumenta que o BDD incorpora e melhora as ideias de muitas dessas metodologias, ajudando assim e tornando a vida da equipe de software mais fácil. Carvalho (2019) complementa que o BDD serve para aperfeiçoar o desenvolvimento e aproximar todas as partes envolvidas no processo de criação, pois a estrutura do BDD é baseada em uma linguagem comum entre todos os integrantes do time.

Dessa forma, Arcanjo (2015) conclui dizendo que o BDD é um conjunto de práticas de engenharia de software projetado para ajudar as equipes a construir e entregar mais rápido software de alta qualidade.

Cucumber (2020) define como essencialmente, a atividade diária do BDD em um processo iterativo de três etapas:

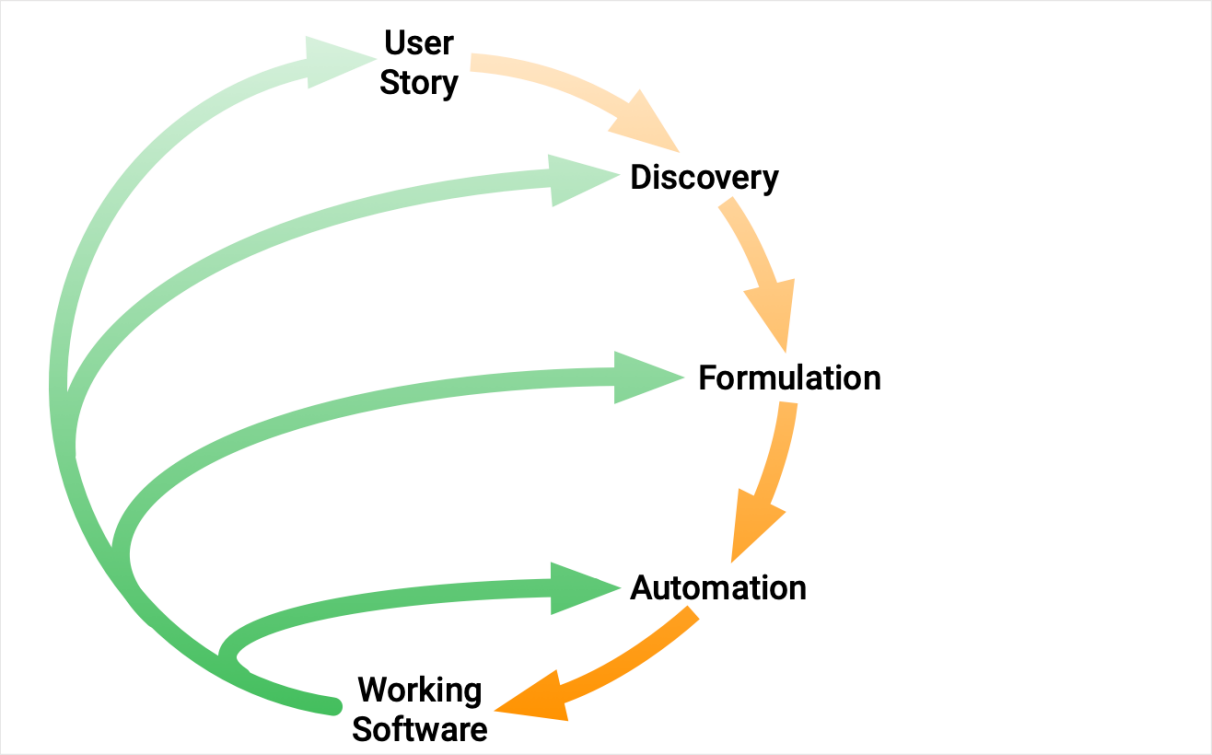
- Primeiro, faça uma pequena alteração no sistema - uma História do usuário - e fale sobre exemplos concretos da nova funcionalidade para explorar, descobrir e concordar com os detalhes do que se espera que seja feito.
- Em seguida, documente esses exemplos de uma maneira que possa ser automatizada e verifique se há acordo.

- Por fim, implemente o comportamento descrito por cada exemplo documentado, começando com um teste automatizado para orientar o desenvolvimento do código.

Esses itens objetivam tornar cada alteração pequena e iterar rapidamente, subindo de nível cada vez que precisar de mais informações. Cada vez que automatizar e implementar um novo exemplo, será adicionado algo valioso ao sistema que estará pronto para responder aos comentários. Prática denominada de descoberta, formulação e automação. (Cucumber, 2020).

Conforme ilustra a imagem a seguir:

Figura 2 - Descoberta, Formulação e Automação

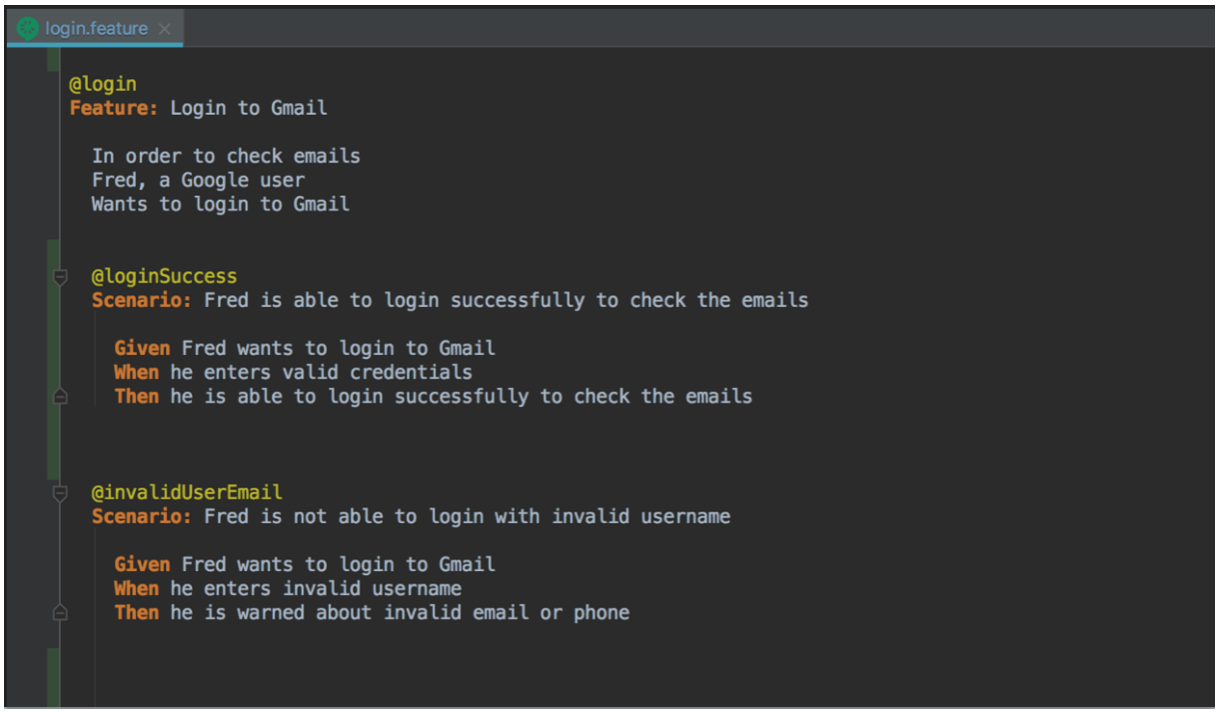


Fonte: Cucumber, 2020

A escrita em BDD geralmente são expressos usando “Cenário” (Scenario), “Dado” (Given), “Quando” (When) e “Então” (Then) (Arcanjo, 2015). No entanto, Carvalho, 2019 ressalva que não é preciso seguir estas normas, mas North recomenda que seja estabelecido um padrão para a escrita, para que a comunicação entre as partes seja mantida.

Segue abaixo o modelo de escrita em BDD:

Figura 3 - Exemplo escrita BDD



Fonte: Hackernoon, 2020

3 METODOLOGIA DO TRABALHO

Este capítulo tem como objetivo apresentar a metodologia utilizada e a forma que o trabalho será aplicado. Primeiramente é caracterizado o tipo de pesquisa em que o trabalho se enquadra. A seguir estão descritas as etapas do desenvolvimento do trabalho. E por fim, o capítulo termina descrevendo as delimitações da pesquisa.

Segundo Motta, et al (2013), a palavra método vem do grego *methodos* que é composta de outras duas palavras, a *metá* (que significa "através de" ou "por meio de") e a palavra *hodós* (que significa "via" ou "caminho"). Ao juntarmos as duas palavras temos um significado de algo como "através do caminho".

Já metodologia descreve os procedimentos metodológicos da pesquisa, ou seja, os recursos que foram utilizados para coletar informações na busca por respostas para o problema. (MOTTA, et al, 2013, pág 60).

3.1 TIPO DE PESQUISA

Para definir o tipo de pesquisa deste trabalho, os autores se basearam nas afirmações de Gil (2002, pág. 17), quem exclama que as pesquisas podem ser determinadas "como o procedimento racional e sistemático que tem como objetivo proporcionar respostas aos problemas que são propostos". Já, conforme o que explicam Silva e Menezes (2005, pág. 20), pesquisa "é um conjunto de ações, propostas para encontrar a solução para um problema, que tem por base procedimentos racionais e sistemáticos. A pesquisa é realizada quando se tem um problema e não se tem informações para solucioná-lo".

Os autores deste trabalho definem o mesmo como uma pesquisa qualitativa, ao analisar o tipo de abordagem utilizada. Nesse tipo de pesquisa os pesquisadores são também objetos de sua pesquisa e o desenvolvimento não segue uma linha racional, pois o conhecimento de quem o desenvolve é parcial e/ou limitado. Mas o importante na pesquisa qualitativa é que ela produza novas informações sobre o assunto, informações essas que sejam aprofundadas e ilustrativas (DESLAURIERS, 1991, p. 58).

Quanto a natureza do trabalho foi determinada pelos autores como uma pesquisa aplicada, por possuir uma aplicação prática e conter um objetivo disposto a solucionar um problema específico. Já analisando os objetivos desta pesquisa, pode-se concluir que ela é classificada como uma pesquisa descritiva, por conter em seus objetivos a descrição de ferramentas e frameworks que auxiliam na automação de testes. Além disso, outro objetivo é a análise e descrição do processo de automação de testes da empresa Softplan, por isso, essa definição de pesquisa descritiva, se aplica. (GIL, 2002, p. 42).

Silva e Menezes (2005) afirmam que uma pesquisa pode ser classificada conforme o ponto de vista dos procedimentos técnicos, portanto esta pesquisa se enquadra no tipo de pesquisa bibliográfica e estudo de caso. Segundo Menezes e Silva (2005, p. 21), a pesquisa bibliográfica é "[...] elaborada a partir de material já publicado, constituído principalmente de livros, artigos de periódicos e, atualmente, com material disponibilizado na Internet." Eles ainda afirmam que a pesquisa, para ser considerada como um estudo de caso, ela "[...] envolve o estudo profundo e exaustivo de um ou poucos objetos de maneira que se permita o seu amplo e detalhado conhecimento". Dessa forma, este trabalho é também um estudo de caso, visto que possui um estudo aprofundado sobre a forma com que a Softplan desenvolve a automação de testes em um dos projetos de sua empresa, além de toda a pesquisa em relação ao assunto que este trabalho está desenvolvendo.

3.2 ETAPAS DO DESENVOLVIMENTO DA PESQUISA

Segundo Rauen (2015, p.562) um estudo de caso possui 3 fases: a) fase exploratória, b) coleta de dados e c) a "fase de análise e interpretação sistemática dos achados com elaboração do relatório. Destaque-se que estas três fases se interpõem, sendo difícil delimitá-las com precisão".

Assim, seguindo as indicações da literatura o processo de desenvolvimento deste trabalho foi dividido nas seguintes etapas:

- Pesquisa bibliográfica e contatos iniciais na empresa (fase exploratória).
- Coleta de dados na empresa (fase de coleta de dados).

Fazendo parte da etapa de elaboração do relatório com a análise e interpretação dos dados constam as seguintes etapas:

- Caracterização da empresa – Produtos desenvolvidos.
- Caracterização do projeto (descrição de onde foi aplicada a automação).
- Descrição do processo de desenvolvimento e testes antes da automação.
- Descrição do processo de desenvolvimento no projeto onde é feita a automação.
- Descrição da automação de testes, processo, ferramentas e tecnologias utilizadas.
- Análise da automação à luz da teoria estudada na revisão da literatura (capítulo 2).
- Elaboração de conclusões.

3.3 DELIMITAÇÕES DO TRABALHO

O presente estudo consiste na descrição de um estudo de caso em automação de testes na empresa Softplan. Após a descrição do processo e ferramentas utilizadas, será feita uma análise com pontos a melhorar. A implantação dessas melhorias na empresa citada não faz parte deste trabalho. A implantação das mesmas melhorias em qualquer outra empresa ou organização não faz parte deste trabalho. A implantação de automação de testes na empresa selecionada para o estudo de caso, ou para qualquer outra empresa ou organização não faz parte do desenvolvimento desta pesquisa. A pesquisa não tem como objetivo, explicar conceitos e configurações necessárias para a utilização das ferramentas e frameworks utilizados neste trabalho. Os autores não têm como objetivo criar nenhum tipo de guia para automação de testes, nem para implantação de testes automatizados em empresas que ainda não os aplicam.

4 DESENVOLVIMENTO

Neste capítulo, é apresentado o estudo de caso da empresa Softplan. O estudo tem como objetivo principal apresentar como foi realizado o processo de automação de testes.

4.1 DESCRIÇÃO DA EMPRESA

O estudo de caso realizado na empresa Softplan, que segundo site da própria organização (SOFTPLAN, 2019) é uma das maiores empresas de software do país, com cerca de 1,9 mil colaboradores e mais de 3500 clientes. Atua há 29 anos no desenvolvimento de softwares de gestão empresarial e gestão pública e com presença internacional. Desenvolve soluções corporativas para segmentos específicos de negócios, com foco nas seguintes áreas:

4.1.1 Justiça

Promovendo uma justiça mais célere, transparente e inovadora a softplan é protagonista na implantação do processo digital da justiça brasileira com o uso de inteligência artificial, analytics, computação cognitiva, machine learning e big data como parte do trabalho. Além de estar à frente da transformação digital na justiça brasileira, também oferece soluções inovadoras. Nessa incansável busca pelas melhores e mais novas tecnologias a transformou em uma das maiores Lawtechs da América Latina.

As soluções desenvolvidas está presente nos tribunais de justiças, advocacia privada, advocacia pública, departamento jurídico e ministérios públicos (SOFTPLAN, 2019).

4.1.2 Administração pública

As soluções usam a tecnologia à serviço do governo e do cidadão, levando transparência e eficiência aos órgãos públicos e sociedade. Eliminar o papel e tornar os processos mais ágeis é um grande desafio em todas as esferas do setor público. Por isso, é fundamental contar com quem tem experiência na condução desse caminho. A Softplan é, portanto, especialista em fazer transformação digital acontecer na gestão pública. Prova disso é que foi retirado o papel do maior Tribunal de Justiça do mundo em volume de processos, o TJSP.

Suas soluções apoiam os gestores públicos no aumento da eficiência, na otimização dos recursos e na melhoria constante dos serviços prestados ao cidadão. Soluções que afetam na fiscalização, gestão pública eficiente, infraestrutura rodoviária, obras públicas e projetos com financiamento externo (SOFTPLAN, 2019).

4.1.3 Indústria da construção

Especialista e há mais de 29 anos a Softplan propõe a entender o complexo mundo da indústria da construção, do canteiro ao escritório. Por isso muito mais que um software, a Softplan possui uma plataforma para a indústria da construção.

O principal diferencial é a paixão por tecnologia da construção. É essa paixão que aguça o olhar para o que ainda será tendência e traz o que é novidade para dentro de casa dessa forma evoluí rapidamente no Ecossistema da Construção.

A solução conecta com as melhores ferramentas e conteúdo da indústria da construção e oferece um ecossistema com o que há de mais moderno e inteligente no mercado. A missão é ser um ecossistema de soluções para a transformação da indústria da construção, e são convidadas empresas, sejam como clientes ou parceiros tecnológicos, a fazerem parte deste movimento crescente (SOFTPLAN, 2019).

4.1.4 Saúde

Com objetivo de transformar a saúde com tecnologia, a Softplan resolve problemas relevantes e comuns ao ecossistema de saúde, de modo a torná-lo mais saudável, ágil, sustentável e humano. O paciente é o maior impactado em todo o processo e precisa estar no centro da atenção ao cuidado. Incentivando mudanças para que as operadoras de saúde possam, através da análise de dados, gerar inteligência em ações de melhoria em processos, promoção à saúde e medicina preventiva.

O portfólio é composto pelas soluções Dictas e BIO. O Dictas entrega insights para ações efetivas na gestão da saúde. O BIO é uma solução completa de coordenação de cuidados assistido por uma equipe médica para entregar saúde de forma contínua (SOFTPLAN, 2019).

4.2 CARACTERIZAÇÃO DO PROJETO

O objeto de estudo de caso será feito na área de Gestão Pública, mais especificamente no sistema Solar BPM.

O Solar BPM é uma plataforma que realiza a gestão dos processos digitais e automatiza as rotinas de trabalho possibilitando eficiência à máquina pública. Elimina as pilhas de papel e o tempo com tramitações desnecessárias, através o uso de ferramentas que gerenciam todo o rito processual administrativo, a gestão arquivística e documental em uma única solução.

4.3 PROCESSO DE DESENVOLVIMENTO E TESTES

A equipe de desenvolvimento objeto de estudo, consistia em 19 pessoas distribuídas nas seguintes áreas:

- 11 desenvolvedores
- 4 analistas de sistemas

- 3 testes
- 1 coordenador
- 1 gestor de projetos

4.3.1 Processo de Desenvolvimento

O processo utilizado pelo setor é o modelo cascata. No entanto, esse não é um processo utilizado de forma universal pela empresa, pois cada área de negócio possui diversos sistemas com clientes diferentes e as equipes se organizam da forma que lhes servirem melhor.

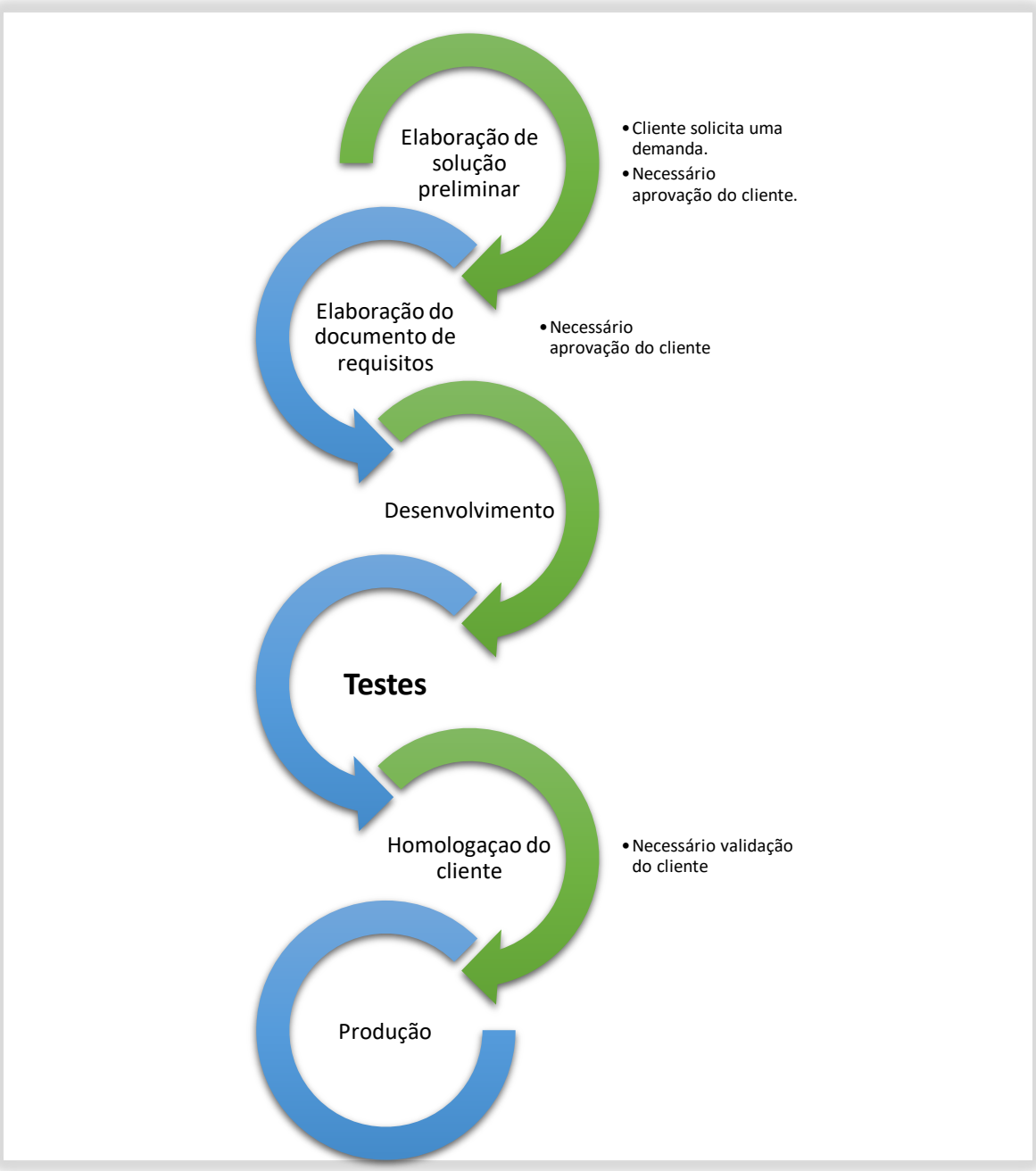
Conforme PRESMAN (2016, p. 41-42):

> O modelo cascata, algumas vezes chamado ciclo de vida clássico, sugere uma abordagem sequencial e sistemática para o desenvolvimento de software, começando com a especificação dos requisitos do cliente, avançando pelas fases de planejamento, modelagem, construção e disponibilização, e culminando no suporte contínuo do software concluído. O modelo cascata é o paradigma mais antigo da engenharia de software. Entretanto, ao longo das últimas quatro décadas, as críticas a este modelo de processo fizeram até mesmo seus mais árduos defensores questionarem sua eficácia.

É muito difícil uma empresa aplicar todos os conceitos de uma metodologia de desenvolvimento, por isso, entende-se que utilizam de metodologias hibridas, onde existem misturas de conceitos.

Abaixo segue uma exemplificação da metodologia de desenvolvimento mais utilizado pela Softplan, que é o modelo de cascata:

Figura 4 - Processo de desenvolvimento



Fonte: Autores, 2020

O modelo acima é uma representação do que ocorre no projeto da Softplan considerado.

4.3.1.1 Elaboração da solução preliminar

Quando o cliente sente uma necessidade, o mesmo solicita uma demanda para a empresa, a qual, é direcionada para a equipe responsável pelo sistema. Com a chegada da demanda, é iniciada uma análise preliminar pelos responsáveis do projeto, entre eles, o analista de sistemas, arquiteto e DBA (*Database administrator*). Essa análise serve para verificar o quanto a demanda afeta o sistema. Dessa análise é gerada uma solução viável com uma estimativa de faixa de custos em pontos por função (APF). Essa solução preliminar precisa ser aprovada pelo cliente para seguir para a próxima etapa.

4.3.1.1.1 Análise de Pontos por função

A equipe alvo desse estudo utiliza-se da análise de pontos por função para avaliar uma estimativa de faixa de custos. A APF trata-se de uma técnica que permite medir a funcionalidade de um software ou aplicativo, sob a visão do usuário, e a partir da descrição dos requisitos do usuário. Ela é utilizada para medir a funcionalidade dos sistemas independentemente da tecnologia, além de ser compreensível para o pessoal não técnico. (HÜBNER, 2020).

Os autores Vazquez, Simões e Albert (2012, pág 31) complementa que a APF busca medir o que software faz, e não como ele foi construído e que o processo de medição, baseia-se em uma avaliação padronizada dos requisitos lógicos do usuário. Além disso, o autor destaca que os pontos por função não medem diretamente esforço, produtividade ou custo. É exclusivamente uma medida de tamanho funcional do software. Este tamanho em conjunto com outras variáveis é que pode ser usado para derivar produtividade.

4.3.1.2 Elaboração do documento de requisitos

O analista inicia a elaboração do documento de requisitos elencando todo o escopo de alteração, bem como os requisitos funcionais e não-funcionais, precificação final do projeto, além de gerar protótipos de telas para que seja aprovado pelo cliente. Somente após aprovação é iniciado o desenvolvimento.

4.3.1.3 Desenvolvimento

A equipe de desenvolvimento analisa a demanda que foi recebida, verifica o contexto do cliente, estuda o projeto como um todo (arquitetura de sistema e tecnologias), para poder definir qual a melhor forma de implementar a solução. A partir disso, é realizada uma divisão entre os integrantes da equipe, para que pequenas entregas sejam passadas a equipe de testes.

4.3.1.4 Testes

Ao receber uma demanda, a equipe de testes analisa a aplicação das melhores técnicas de teste de software. Mais uma vez, o contexto do cliente, a tecnologia utilizada na implementação, além do tipo de solução implementada, tem influência no teste que será realizado. Se a demanda pertencer a algum tipo de solução recorrente, que seja identificada pela equipe de testes, pode haver algum processo específico para testar como por exemplo, a utilização de *check-lists.* Nessa etapa podem ser identificadas questões relacionadas aos requisitos ou de implementação, que pode retornar a demanda para algumas das etapas anteriores. Se for julgada como aprovada, a solução segue para homologação com o cliente.

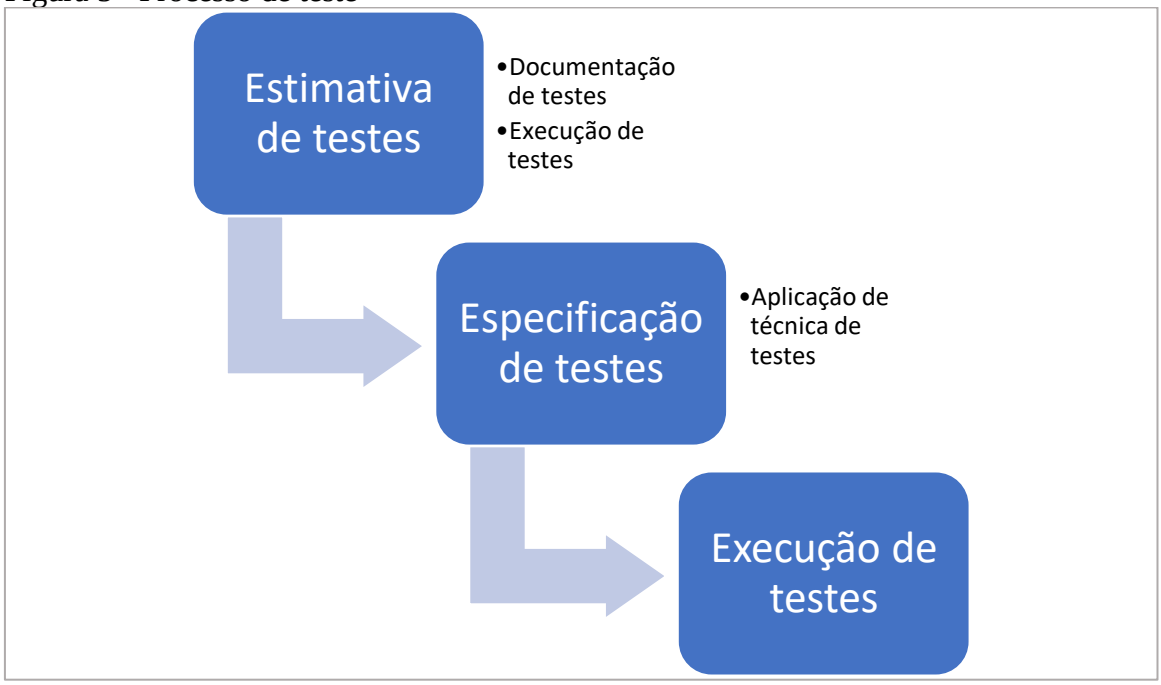
4.3.1.5 Homologação do cliente

Na etapa de homologação, é liberada uma versão no qual, é apresentado ao cliente toda a implementação realizada nas etapas anteriores. O cliente por sua vez, realiza testes de conformidade, e compara com os requisitos solicitados no início do processo. Caso entenda que existe alguma inconformidade, essa demanda mais uma vez pode retornar para algum dos processos anteriores.

4.3.2 Processo de Testes

Todos os testes realizados pela equipe alvo do estudo passava pelos seguintes passos a seguir:

Figura 5 - Processo de teste



Fonte: Autores, 2020

4.3.2.1 Estimativa de testes

Nessa etapa eram realizados basicamente dois tipos de estimativas:

- Estimativa de documentação de testes.
- Estimativa de execução de testes.

Primeiramente era realizado a escolha do caso de teste a ser implementado. Isso depende de todas as variáveis citadas anteriormente, como o tipo de demanda, a tecnologia utilizada na implementação, o histórico de cliente, entre outros.

Depois, era gerada uma estimativa do tempo necessário para realizar o teste. Essas estimativas seguem padrões específicos, que auxiliam o analista de testes a avaliar quanto tempo uma aplicação de teste pode levar. O próximo passo é criar toda a documentação necessária para realizar o teste.

4.3.2.2 Especificação de testes

Neste momento são mapeados todos os itens necessários para a criação da especificação de testes. Os itens são:

- Requisitos funcionais: Representa as funcionalidades do sistema, ou seja, as funções que o sistema deverá realizar.
- Requisitos não funcionais: Define as exigências técnicas da aplicação.
- Regras de negócio: Define regras de como o sistema deve se comportar, ou seja, como o sistema deve agir em determinada situação.
- Regras de tela: Define como um elemento de tela deve se comportar.
- Mensagens: Define qual a descrição de mensagem a ser exibido ao usuário mediante uma validação.
- Fluxos de exceção: Define cenários alternativos a função principal do sistema.

Com base nos itens mensurados, o analista de teste se vale de técnicas para escrever os cenários da melhor forma possível.

4.3.2.3 Execução de testes

Esse é o momento onde todos os artefatos de especificação de testes devem estar prontos para execução, ou seja, aqui o analista deve obter todos os acessos necessários, disponibilização de ferramentas a serem utilizadas nos testes, informações, gráficos, tudo que seja intrínseco para realizar o teste. Este é o ponto onde o analista dá início ao teste em si.

4.4 PROCESSO DE AUTOMAÇÃO.

Neste capítulo será apresentado a forma como foi realizado o piloto de automação de testes na equipe alvo desse estudo de caso.

4.4.1 Estudo das ferramentas

Primeiramente, para que todo o processo de automação fosse possível, foi necessário realizar um estudo minucioso das ferramentas disponíveis no mercado, bem como a relação de preços entre elas, além das tecnologias utilizadas por cada uma. Esse estudo foi realizado com o apoio da coordenação da empresa. Conforme visto no capítulo 2, a automação está bem consolidada no ambiente de testes, e isso faz com que empresas, como a Softplan, possam se interessar pelos benefícios que testes automatizados podem trazer.

4.4.2 Elaboração das POCs

Para que o projeto pudesse ser iniciado, foram realizadas algumas POCs (provas de conceitos). A realização de uma POC é a implementação de uma versão simplificada do framework a ser utilizado. Esse processo foi utilizado na perspectiva de quais ferramentas seriam mais apropriadas para a necessidade da equipe. Nesse caso, as POCs foram implementadas especificamente nos *frameworks* estudados para aderirem ou não ao processo de automação de testes.

Dentre as ferramentas, estavam Robot, Cucumber e Ranorex. Durante a implementação da POC foi possível constatar as principais características das ferramentas, que serão descritas a seguir:

4.4.2.1 Robot Framework

O Robot Framework é uma estrutura de automação extensível e orientada a palavra-chave baseada em Python para testes de aceitação, além disso é uma ferramenta opensource lançado sob a Apache Licence 2.0. Apesar de ser orientada a palavra-chave, também é possível escrever no formato BDD, como também possui vasto ecossistema composto por várias bibliotecas e ferramentas genéricas desenvolvidos como projetos separados. Podendo também criar bibliotecas para compartilhar com a comunidade. Em geral robot e cucumber pode fazer a mesma coisa, mas de modos diferentes.

Figura 6- Test case - robot framework

```
*** Settings ***
Documentation     A test suite with a single test for valid login.
...
...               This test has a workflow that is created using keywords in
...               the imported resource file.
Resource          resource.txt

*** Test Cases ***
Valid Login
    Open Browser To Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

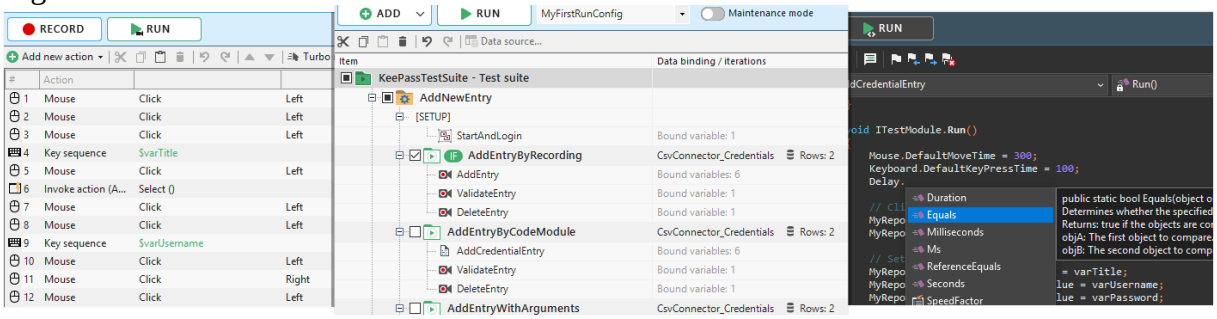
Fonte: Robot framework, 2020

4.4.2.2 Ranorex

O Ranonex é uma ferramenta comercial de automação de testes, ou seja, seu uso depende do pagamento da licença. A ferramenta não usa nenhuma linguagem de script especifica, sendo os testes construídos na própria plataforma .NET Microsoft. A interface de reprodução e registro do Ranorex permite automatizar o teste da interface do usuário gravando suas ações. Os principais pontos que destacamos:

- **Custo**: Este foi o fator de maior relevância na decisão de não a usar, pois é uma ferramenta licenciada, cujo investimento mínimo para obtê-la é de USD $5590.
- Confiabilidade: Grandes empresas utilizam a ferramenta para automação como DELL, Adidas, Cisco, Siemens, entre outras.
- **Curva de aprendizado**: A interface é amigável. No entanto é um pouco trabalhoso para quem nunca lidou com a ferramenta dificultando em como começar e estruturar os testes, fazendo com que seja mais dependente do suporte da ferramenta.
- **Comunidade**: A base do Ranorex é muito menor comparada com as outras que são opensource, dificultando encontrar soluções de problemas que possa vir a ter.
- **Uso**: O uso da ferramenta seria de conhecimento e uso exclusivo da equipe de testes, deixando uma forte dependência de conhecimentos em uma pessoa. Podendo gerar muito mais custo caso essa pessoa saia da empresa. Além disso, não teria o suporte da própria equipe, principalmente dos desenvolvedores que comparadas as outras ferramentas podem auxiliar nos problemas com os códigos visto que são programadas.

Figura 7 - Interface ranorex



Fonte: Renorex, 2020

4.4.2.3 Cucumber

Após avaliação foi considerada mais adequada a ferramenta Cucumber, primeiramente pela afinidade dos frameworks complementares ao cucumber, como também não possuir a necessidade de criar uma ambiente próprio só para automatizar e por ser compatível com a necessidade da equipe, além do fato de ser uma ferramenta *open-source,* por ser bem avaliada no mercado e possuir uma grande comunidade foram fatores determinantes para escolha. Além disso, a ferramenta possui um rápido processo de aprendizagem, o que diminui a implementação de automação em uma equipe iniciante. As ferramentas que foram selecionadas serão apresentadas logo a seguir, junto com a descrição do processo de automação.

4.4.3 Apresentação para a equipe

Um dos pontos cruciais do processo, foi a apresentação do projeto de automação para os gestores e o restante da equipe, pois sem a aprovação de implementação, todo o esforço realizado até o momento não teria nenhuma validade. Felizmente toda a equipe e gestores mostraram uma reposta positiva em relação a proposta apresentada. Somente a partir dessa aprovação foi que o processo de automação realmente teve início na equipe.

4.4.4 Automação

Inicialmente foi gerado um mapeamento de todos os cenários e casos de testes utilizados pela equipe. A partir disso pode-se priorizar quais seriam automatizados no projeto piloto e quais cenários seriam automatizados posteriormente. Uma grande vantagem desse passo foi que também pode-se definir quais cenários não seriam automatizados em nenhum momento, isto é,

são cenários que sua execução exige permanecer manual. Após isso foi definido a estrutura BDD além da definição da estrutura do teste dentro do framework.

Segundo Cucumber (2020), esta é uma ferramenta que auxilia no processo de desenvolvimento de software, visando melhorar a qualidade do software e reduzir os custos de manutenção.

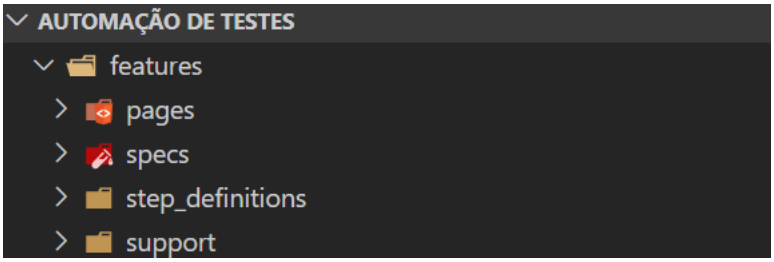
Os fatores que levaram a escolha do framework se alicerçaram na possibilidade da escrita em linguagem simples, ou seja, em texto em português cuja descrição se dá em linguagem de negócio. Dessa forma, o Cucumber e BDD contribuíram para:

- Documentação, levando em conta que a aplicação não possui a documentação do legado;
- Redução do esforço para manter as especificações;
- A escrita de cenários, que pode ser entendida por qualquer usuário leigo;
- Fácil manutenção nos *templates* de testes, podendo qualquer membro da equipe colaborar;
- Documentação dos requisitos sincronizados;
- E a própria automação de testes;

A ferramenta permite a execução dos cenários de testes escritos em textos em português (não em linguagem de programação), ou seja, o BDD. Os cenários são inseridos em arquivos *feacture,* podendo abranger vários cenários conforme estrutura definida das funcionalidades do sistema. O Cucumber interpreta esses cenários na linguagem de programação especificada e utilizada o Selenium para manipular os testes no navegador.

O Cucumber se organiza por *features* e basicamente se estrutura em quatro partes, conforme imagem abaixo:

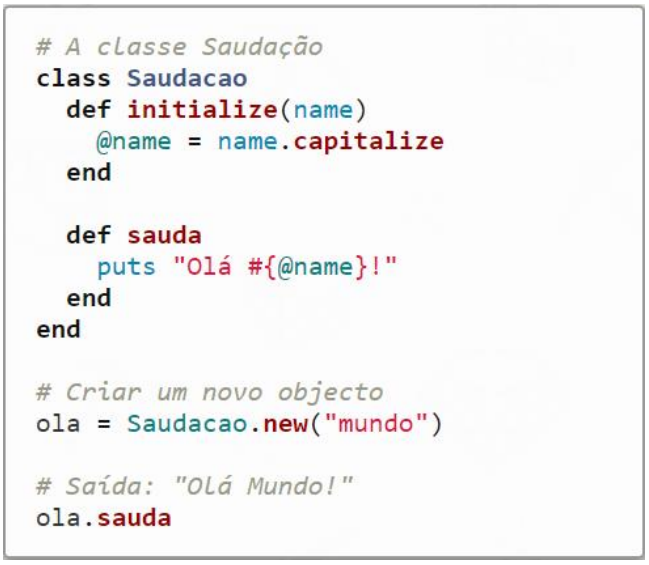
Figura 8 - Estrutura cucumber



Fonte: Autores, 2020

Antes de abordar cada pasta, será necessário falar sobre a linguagem utilizada com o Cucumber, o Ruby.

Figura 9 - Exemplo sintaxe linguagem Ruby



```ruby
# A classe Saudação
class Saudacao
  def initialize(name)
    @name = name.capitalize
  end

  def sauda
    puts "Olá #{@name}!"
  end
end

# Criar um novo objecto
ola = Saudacao.new("mundo")

# Saída: "Olá Mundo!"
ola.sauda
```

Fonte: Ruby, 2020

A linguagem Ruby está entre os *top* 20 nos índices que medem o crescimento da popularidade de linguagens de programação pelo mundo todo (tais como o índice TIOBE). Muito deste crescimento é atribuído à popularidade de softwares escritos em Ruby, em particular o *framework* de desenvolvimento *web Ruby on Rails* (RUBY, 2020).

Aplicativos como *Basecamp, GitHub, Shopify, Airbnb, Twitch, Soundcloud, Hulu, Zendesk, Square* e *Cookpad* foram desenvolvidos com Ruby.

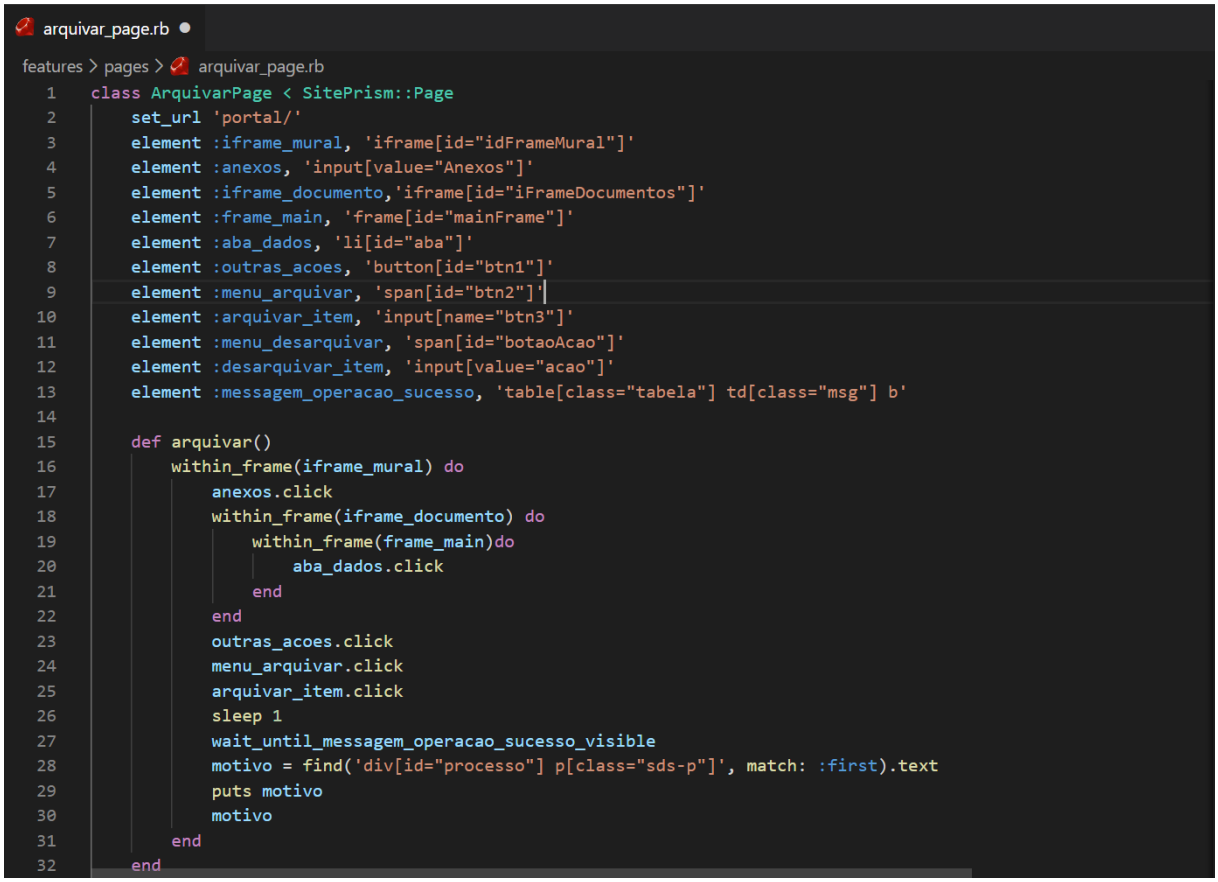
Conforme MasterTech (2018):

> Assim como o PHP, Ruby é outra linguagem de script que os desenvolvedores usam para desenvolvimento web e forma a base do *framework Ruby on Rails.* Quando se trata de desenvolvimento de aplicativos móveis baseados na *web*, o Ruby se mostra uma linguagem de programação mais amigável, especialmente para iniciantes e tem uma comunidade de usuários forte e generosa pronta para ajudá-lo quando você encontrar problemas súbitos.

Portanto, a escolha da linguagem supracitada na automação de testes se deu, pelo fato do *ruby* ser uma linguagem amigável para iniciantes. Além do que o *cucumber* foi desenvolvido com Ruby.

Pages: Nesta pasta, são implementados os métodos dos testes e elencados todos os elementos das telas que serão utilizados na programação da automação.

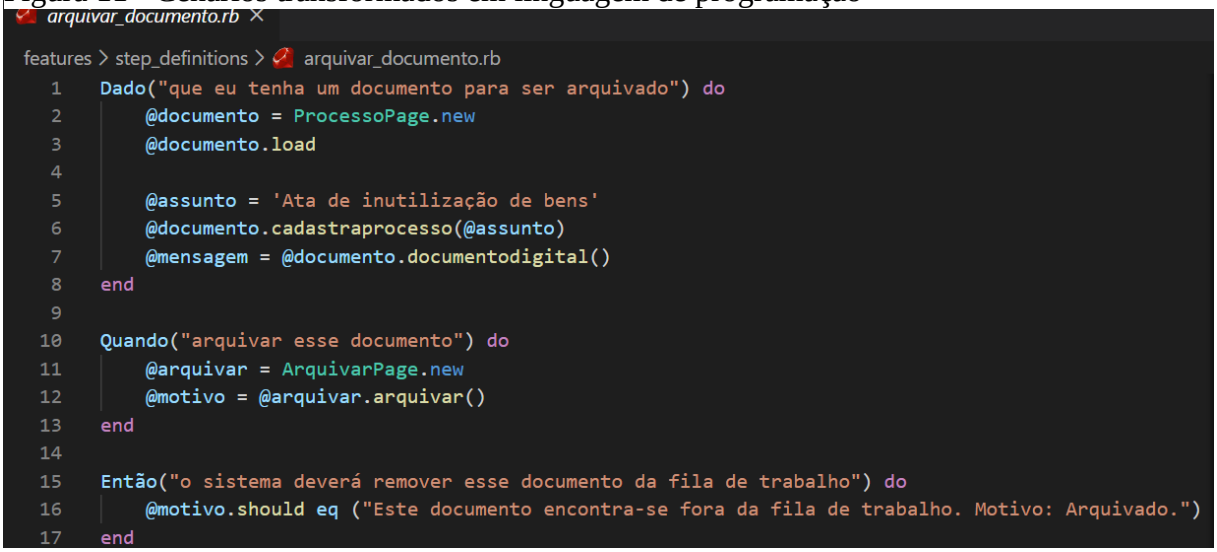
Figura 10 - Implementação das pages em ruby



Fonte**:** Autores, 2020

***Step_definitions*:** Nesta pasta, os cenários que foram definidos são transformados e interpretados em linguagem de programação.

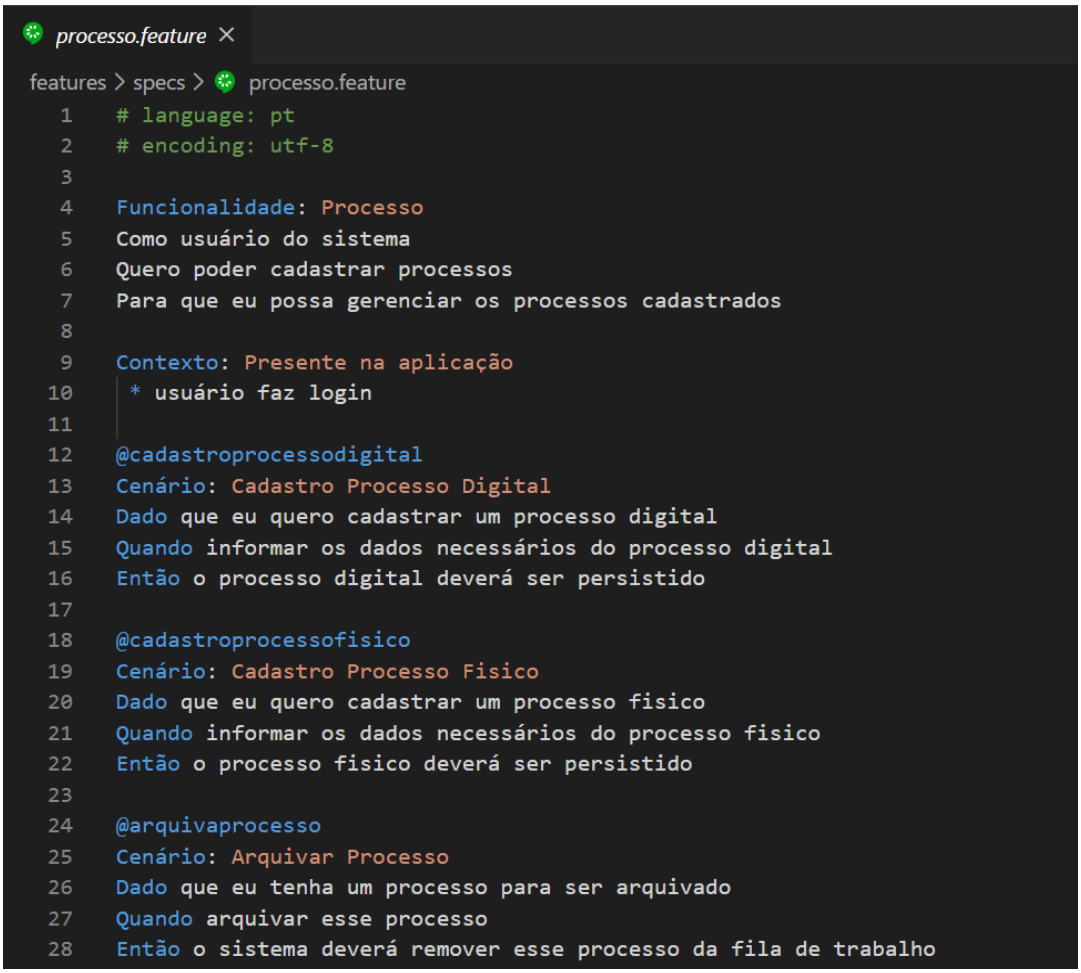
Figura 11 - Cenários transformados em linguagem de programação



Fonte: Autores, 2020

***Specs*:** Nessa pasta são definidos os cenários da automação em BDD. Abaixo segue um exemplo de como é utilizada a *spec:*

Figura 12 - Cenários em BDD



processo.feature

features > specs > processo.feature

```
# language: pt
# encoding: utf-8

Funcionalidade: Processo
Como usuário do sistema
Quero poder cadastrar processos
Para que eu possa gerenciar os processos cadastrados

Contexto: Presente na aplicação
  * usuário faz login

@cadastroprocessodigital
Cenário: Cadastro Processo Digital
Dado que eu quero cadastrar um processo digital
Quando informar os dados necessários do processo digital
Então o processo digital deverá ser persistido

@cadastroprocessofisico
Cenário: Cadastro Processo Fisico
Dado que eu quero cadastrar um processo fisico
Quando informar os dados necessários do processo fisico
Então o processo fisico deverá ser persistido

@arquivaprocesso
Cenário: Arquivar Processo
Dado que eu tenha um processo para ser arquivado
Quando arquivar esse processo
Então o sistema deverá remover esse processo da fila de trabalho
```

Fonte: Autores, 2020

***Support*:** Nesta pasta são definidas as configurações gerais da automação, como o navegador a ser executado, as dependências ou ambiente de execução por exemplo.

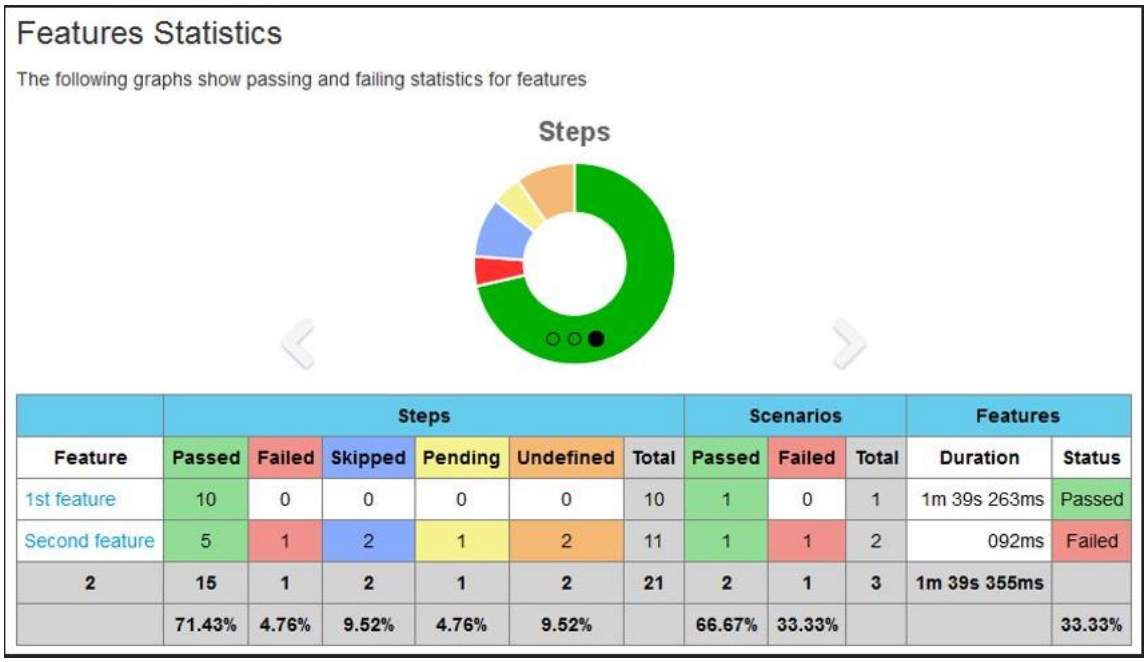
O cucumber conta também com recursos com uso de plug-ins de status report para reproduzir relatórios dos cenários elaborados, exibindo as falhas e aprovações. Já existe alguns relatórios nativos, mas outros podem ser instalados separadamente ou pode criar o seu próprio, vide alguns exemplos abaixo:

Figura 13 - Exemplo de status report 1 - cucumber

	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Arquivamento	24	0	0	0	0	24	6	0	6	3:16.290	Passed
Assinatura	32	0	0	0	0	32	8	0	8	4:32.534	Passed
Cadastros básicos	8	0	0	0	0	8	2	0	2	24.091	Passed

Fonte: Autores, 2020

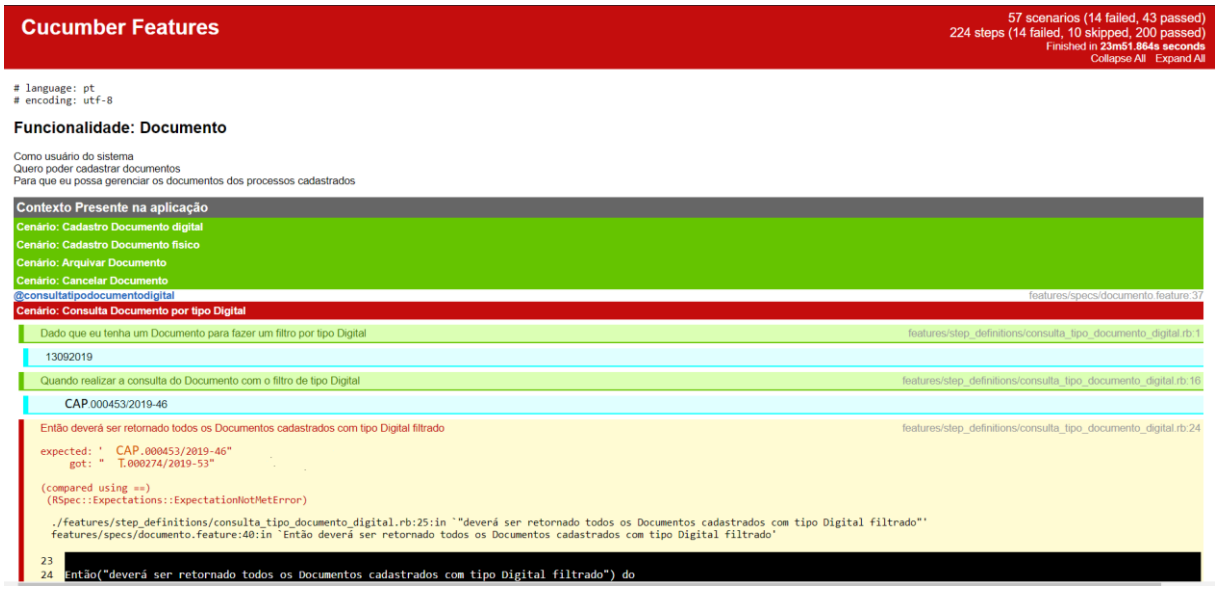
Figura 14 - Exemplo de status report 2 - cucumber



	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
1st feature	10	0	0	0	0	10	1	0	1	1m 39s 263ms	Passed
Second feature	5	1	2	1	2	11	1	1	2	092ms	Failed
2	**15**	**1**	**2**	**1**	**2**	**21**	**2**	**1**	**3**	**1m 39s 355ms**	
	71.43%	**4.76%**	**9.52%**	**4.76%**	**9.52%**		**66.67%**	**33.33%**			**33.33%**

Fonte: Autores, 2020

Figura 15 - Exemplo de status report 3 - cucumber



Fonte**:** Autores**,** 2020

Além disso, utilizamos de forma complementar o ***Rspec***, que segundo Campos (2011) é um framework de testes de unidade, assim como o JUnit. Porém ele vai além de apenas fornecer um conjunto de métodos para os testes, e nos fornece uma DSL para descrever os comportamentos esperados do sistema, ou seja, ao invés, de ter apenas "um monte" de *asserts*, temos "um monte" de *asserts* dentro de exemplos executáveis, a diferença é sútil, mas faz uma boa diferença.

Ademais, também fora utilizada o ***Capybara,*** que conforme Milani (2019) é um Framework de testes de aceitação para aplicações web escrito na linguagem de programação *ruby*. Ele pode imitar ações de usuários reais que interagem com aplicativos baseados na web.

Lara (2017) também reforça as vantagens do capybara ao afirmar que:

> Com ele é possível testar as aplicações simulando as ações que os usuários reais executariam, ou seja, se eu submeto um formulário, a aplicação deverá ser direcionada para a próxima página válida. A partir da utilização de Drivers, o capybara realiza a simulação das ações dos usuários utilizando-se de frameworks que controlam navegadores ou que simulam a execução dos mesmos em memória, tornando mais

> rápida a execução dos testes. Alguns exemplos de ações simuladas seriam: Acessar URLs, preencher campos, clicar, verificar se valores foram apresentados na tela, etc.

Segue abaixo alguns comandos utilizados para simular as ações dos usuários:

Figura 16 - Comandos capybara

Comando	Descrição	Exemplo
visit	Acessar uma URL	visit "https://medium.com/jaguaribetech"
expect(page).to have_content	Verifica se a página possui o texto esperado	expect(page).to have_content ('Cadastro realizado com sucesso')
fill_in	Adiciona um texto a um elemento ou campo	fill_in "login", with: "samyralara"
has_content?	Verifica se um texto está sendo apresentado em um elemento	page.has_content?("Samyra Lara")
visible?	Verifica se um elemento está visível	find(".status").visible?
click_button	Clica em um determinado botão	click_button 'login'

Fonte: Lara, 2018

Conforme supracitado, é um framework escrito na linguagem Ruby. E em sua página (https://www.rubydoc.info/github/jnicklas/capybara) consta toda a documentação dos comandos disponíveis.

4.5 DESCRIÇÃO E ANÁLISE DA IMPLEMENTAÇÃO DA AUTOMAÇÃO

Nesta seção são apresentados alguns dados relacionados aos testes que foram automatizados. Será possível verificar qual foi o ganho de tempo em relação ao processo anterior, quando esses testes eram realizados manualmente, e mostrar esse ganho através de gráficos.

O sistema é fruto de um legado, no qual, evoluções estão sendo realizadas para atualizar as funcionalidades de acordo com as necessidades novas, além de inovar, agregar valor ao

cliente, bem como modernizar. O fato de o sistema ser legado, acarreta que as evoluções são muito sensíveis a impactos conforme complexidade da alteração.

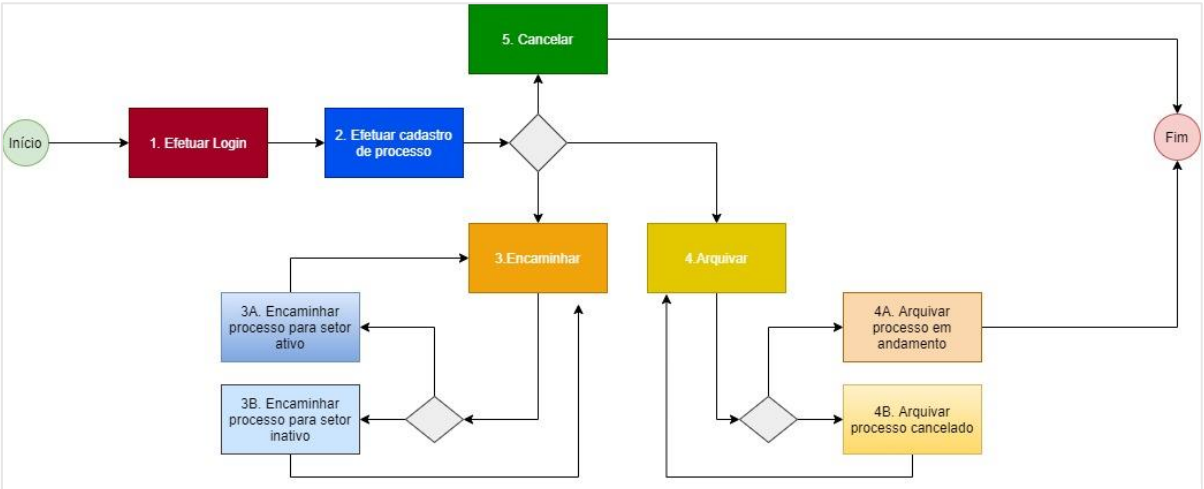
Foram automatizados 73 cenários, cuja estratégia adotada fora cobrir as funcionalidades mais usadas pelos usuários. No entanto, é preciso destacar que neste primeiro momento não seriam cobertos os cenários de exceção e sim o caminho óbvio, para o qual o termo utilizado é “caminho feliz”. É importante destacar que todos os cenários automatizados estão relacionados a testes de regressão.

A implementação da automação foi um desafio animador, visto a facilidade de aprendizado no uso dos frameworks.

Os testes automatizados são executados de forma única, de modo que cada teste é como se fosse uma nova seção no navegador. A linha de implementação seguiu a diretriz, de que os cenários são independentes, assim os mesmos puderam ser executados de forma isolada e reaproveitada. Dessa forma, os testes não ficaram dependentes de um ambiente próprio para automação, visto o que foi e, visto que a criação de ambiente precisa seguir um processo burocrático. Com isso, os testes poderiam ser executados em qualquer ambiente, desde que possuísse no mínimo a massa de dados parametrizados na automação.

Por exemplo, na funcionalidade de “Arquivar um processo”, serão necessários os seguintes passos abaixo, para o sistema poder permitir o arquivamento:

Figura 17 - Arquivar um processo administrativo



Fonte: Autores, 2020

Na imagem acima, todos os passos são passíveis de serem automatizados, dos quais foram implementados apenas os testes referentes aos passos 1, 2, 3A, 3B, 4A, 4B e 5 todos

independentes entre si. Como pode-se observar o cenário 3B e 4B não foram criados, pois são considerados cenários de exceção. Dessa forma para efetuar o cadastro do processo é necessário estar logado no sistema, na automação implementada basta somente chamar o método de login (1) no cadastro de processo (2), para que o login seja realizado antes de iniciar o cadastro de processo, ou seja, não é necessário reimplementar o teste de login no teste de cadastro de processo.

E por fim, para arquivar um processo é necessário estar logado e ter um processo cadastrado, logo, basta chamar o método de cadastro de processo (2), que referencia o login, no teste de arquivar (4A).

Analisando as métricas de tempo de testes, observou-se que os cenários levariam entre 3 a 8 dias de execução manual de testes, dependendo do nível de experiência do analista de teste a efetuar os testes. Levando em conta a complexidade do negócio:

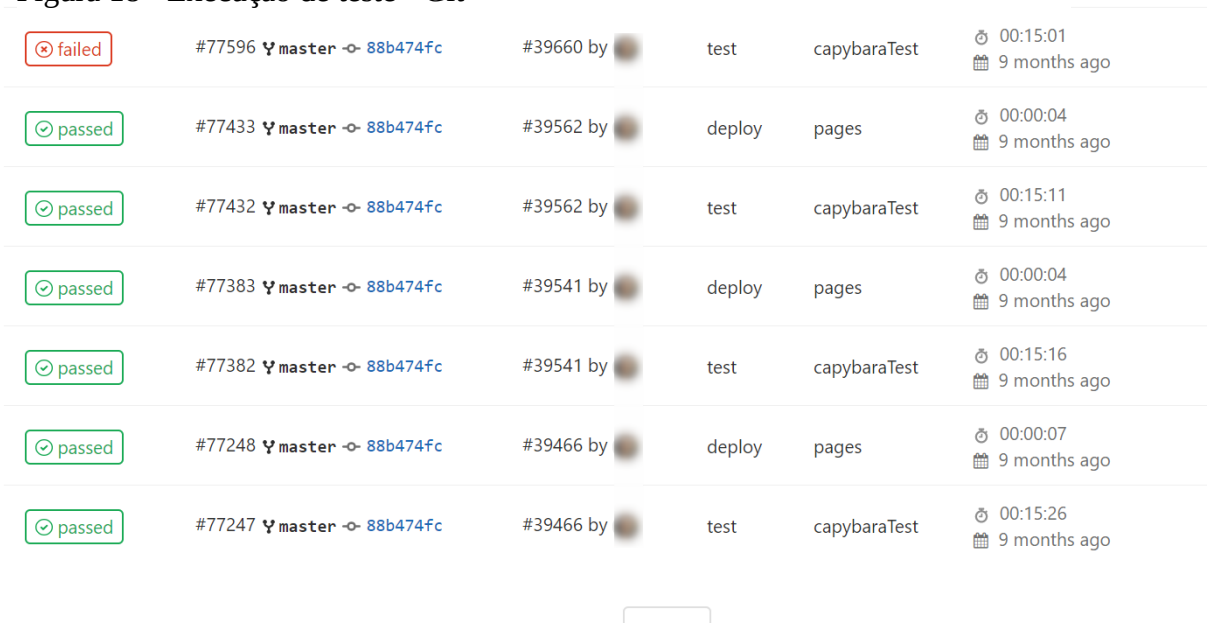
- Um analista de teste júnior, sem conhecimento do negócio poderia levar entre 5 a 8 dias de execução.
- Um analista de teste mais experiente, levaria entre 3 a 4 dias.

Essas métricas e dados citados acima foram coletados a partir do conhecimento e da observação dos próprios analistas de testes com relação ao tempo de execução de cada tarefa, ou seja, são apoiados nas experiências que diferentes analistas da Softplan tiveram ao executar manualmente cada um dos cenários de testes. Consideramos por tanto, os dados relacionados ao tempo de execução dos testes como sendo de conhecimento empíricos.

Um grande benefício foi a possiblidade de inserir os testes automatizados no gitlab, no repositório junto as alterações de desenvolvimento do sistema da empresa. Isso possibilitava efetuar alterações por mais de uma pessoa, acompanhar as alterações, efetuar merges, iniciar as execuções, analisar as falhas e sucessos.

As primeiras execuções já demonstraram serem promissoras, a execução dos testes levou entre 15 a 25 minutos. Como foi mencionada as alterações são sensíveis ao sistema, visto se tratar de um legado que remete a mais de uma década. Os dados dessas execuções se encontram na imagem abaixo extraída do git:

Figura 18 - Execução de teste - Git



Fonte: Autores, 2020

Levando em consideração os dados acima e supondo que seja necessário efetuar testes de regressão no mínimo duas vezes ao mês durante o ano, consideramos o cenário mais otimista: Um analista de teste *pleno/sênior* que levaria 3 dias (24 horas) e um analista de teste *júnior* levaria 5 dias (40 horas) para executar manualmente o teste de regressão. Com isso definido, segue o quadro abaixo mostrando a relação em horas necessárias à equipe com testes manuais desenvolvidas durante o ano.

Quadro 3 - Teste de regressão manual em horas

	1 DIA	1 MÊS	1 ANO
Analista Pleno/Sênior	24	48	576
Analista Junior	40	80	960

Fonte: Autores, 2020

Para esse estudo foi desconsiderado o ganho de experiência do analista de testes *júnior*, desse modo, em apenas um ano de trabalho, o analista *pleno/sênior* necessita de apenas 60% (sessenta por cento) do tempo que leva um analista *júnior*.

A partir deste ponto a equipe então inicia o processo de automatização de testes, onde começa-se a reunir dados em relação ao tempo necessário para treinamento da equipe, tempo de implementação dos testes propriamente ditos, tempo de execução e o acúmulo desses tempos

em um ano. O quadro abaixo traz esses dados referindo-se ao mesmo teste de regressão considerado no quadro anterior, porém agora com a automação implementada.

Quadro 4 - Teste de regressão automatizado

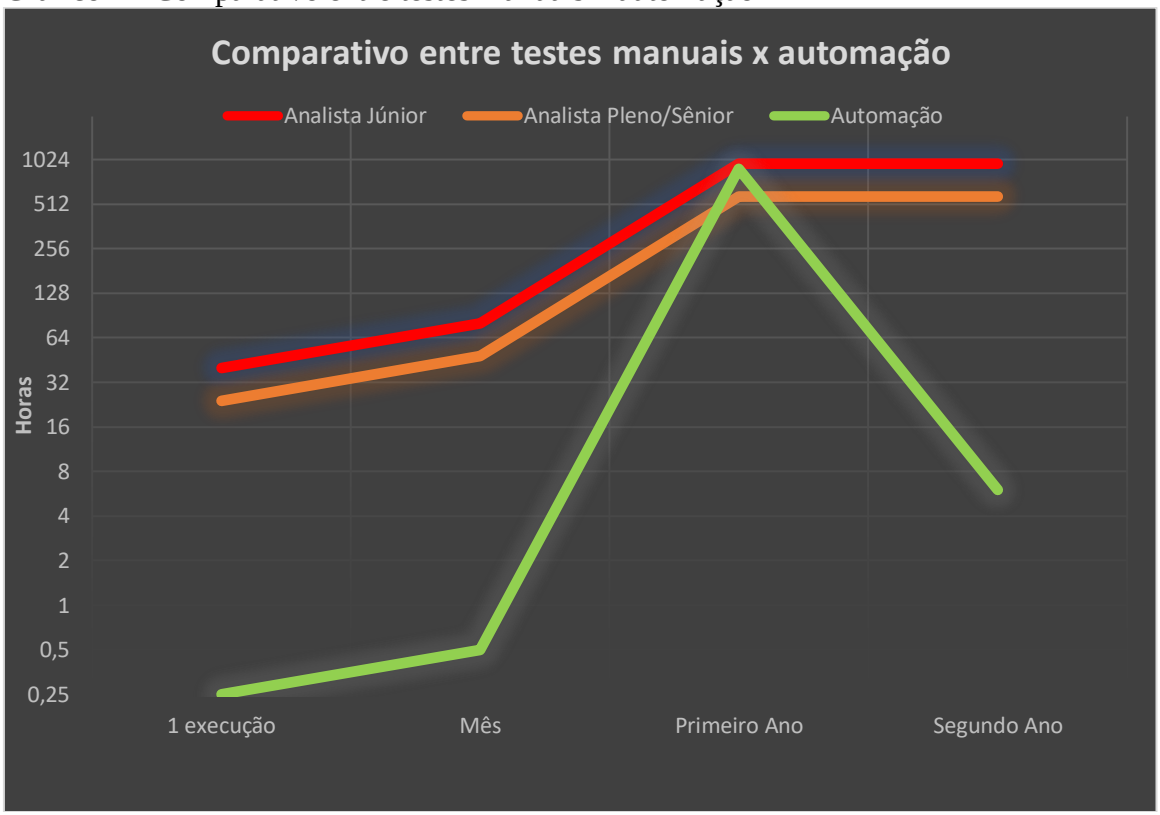
	Tempo investido automação	Tempo por execução (min)	Hora total ano	Hora total ano + Tempo investido automação
1ª ano	• 50 dias treinamento • 60 dias na automatização	15’ a 20’	6 a 8	886 a 888
2ª ano		15’ a 20’	6 a 8	6 a 8

Fonte: Autores, 2020

Com os dados expressos acima, observa-se que no primeiro ano teve um investimento inicial já que não havia alguém com expertise necessária na equipe. Dessa forma, os números foram bem expressivos quando comparados com os testes manuais, de modo que, um analista *pleno/sênior* precisa de menos horas de execução que com a forma automatizado, contudo, o analista júnior gastou um pouco mais de tempo quando comparado com a forma automatizada.

Todavia é imprescindível a comparação com o 1º quadro com os dados do 2º ano do segundo quadro, onde verifica-se uma notória redução de tempo tanto quanto para analista *júnior*, quanto para o *analista pleno/sênior* em relação aos testes automatizados. Para uma melhor visualização dos dados levantados nas duas tabelas foi gerado um gráfico comparando testes manuais e automatizados:

Gráfico 1 - Comparativo entre testes manuais x automação



Fonte: Autores, 2020

Este gráfico denota a importância dos testes automatizados. Com ele pode-se ver a forma como acontece a redução de tempo investido, tanto por uma equipe de testes, quanto por uma equipe de desenvolvimento como um todo, pois a fase de testes está contida pelo desenvolvimento e quando ela sofre uma redução de tempo, o processo de desenvolvimento inteiro é afetado. Outro ponto importante a se destacar sobre este gráfico está quando comparamos uma única execução do teste de regressão. É possível notar que manualmente o tempo necessário fica entre 3 e 5 dias (24 a 40 horas) e automaticamente, esse tempo cai para apenas 15 minutos (0,25 horas).

Conforme destacado no gráfico acima, com a implementação da automação pela equipe de testes, o tempo de atuação da mesma, pode ser reduzido a uma pequena fração do tempo do mesmo teste quando realizado manualmente. Sendo assim, o time de testes assume um papel muito importante na equipe e em todo o processo de desenvolvimento.

Após a análise dos dados e de toda a automação que foi realizada pela equipe, foi gerado um comparativo entre os testes de regressão quando são executados manualmente e automaticamente, considerando apenas a experiência que a equipe obteve:

Quadro 5 - Comparação entre Automação x Testes manuais

Testes manuais	Testes automatizados
Maior tempo de execução	Menor tempo de execução
Apoio humano analítico	Execução mecânica
Menor investimento inicial	Maior investimento inicial
Maior taxa de erros nos testes	Menor taxa de falhas humanas
Retorno a curto prazo	Retorno a médio/longo prazo
Exige menor conhecimento técnico	Exige maior conhecimento técnico
Testes únicos	Reutilização de testes

Fonte: Autores, 2020

É de suma importância enfatizar que o comparativo acima faz referência aos testes de regressão manuais e automatizados pela equipe de testes da Softplan, e que é recomendado que a cada novo tipo de testes automatizado, seja criado um comparativo como esse.

4.6 ANÁLISE DA AUTOMAÇÃO À LUZ DA TEORIA ESTUDADA

Este capítulo dedica-se a levantar alguns pontos do projeto piloto de automação da equipe do Solar BPM da Softplan, que é alvo desse estudo, e realizar uma análise à luz da teoria apresentada no capítulo 2 deste trabalho, com a finalidade de gerar uma lista de recomendações e melhorias, caso seja necessário. Dessa forma atendendo à etapa de análise e interpretação sistemática dos achados em um estudo de caso. Essa etapa pode gerar uma lista de recomendações e melhorias.

4.6.1 Analisando ponto a ponto

A literatura estudada através deste trabalho mostrou alguns pontos essenciais que devem ser levados em consideração no momento que uma empresa/equipe decide iniciar o processo de automação de testes, citados na seção 2.3.

O primeiro ponto a ser levantado é sobre o crescimento elevado de sistemas e aplicações, onde se torna difícil o controle e elaboração de testes manuais. A Softplan possui uma variação

muito grande em relação a sistemas e suas funcionalidades e quantidades diferentes de testes que são aplicadas diariamente, por isso entendemos que esse ponto está umbilicalmente ligado ao processo que houve na empresa.

Já a segunda razão, baseia-se na alta complexidade dos softwares que são desenvolvidos atualmente. Esse ponto também é entendido pelos autores do projeto, como importante e concordante com o que levou a empresa a iniciar a automatização.

A bibliografia apresentada também reporta sobre os *frameworks* mais utilizados atualmente para implementação de teste automatizados. Pode-se verificar que a ferramenta escolhida para auxiliar a empresa na automação está presente nessa lista, além disso, outras ferramentas analisadas como possíveis integrantes ao projeto, também constam na lista apresentada na seção 2.3.1 deste trabalho. Conclui-se, portanto, que a empresa conseguiu buscar uma boa solução para sua necessidade, conforme as diretrizes delineadas pela teoria.

O tipo de teste que foi automatizado pela equipe, faz parte dos testes que a literatura recomenda que sejam automatizados. Isso pode ser verificado quando observamos que até o momento, apenas testes de regressão foram automatizados. Na seção 2.2.2 observa-se que testes de regressão fazem parte dos tipos de testes indicados a automação. Também é evidenciado quando observarmos que no próprio capítulo de introdução deste trabalho, é afirmado que testes de regressão, quando são repetidos inúmeras vezes, são candidatos a automação. Por esses motivos, avaliamos como uma boa escolha da empresa, iniciar o processo de automação pelos testes de regressão.

4.6.2 Lista de Recomendações e Melhorias

Com base no material estudado, dados e informações apresentadas neste trabalho e no contexto da equipe e sistema objeto do estudo, foi gerada pelos autores uma lista com recomendações e melhorias no processo de automatização, bem como nos processos da equipe de testes:

- Alocar um analista responsável pela automação *full-time:*

Essa foi uma grande necessidade que os autores identificaram. Até então, o único profissional que é responsável pela automação na equipe, ainda emprega parte do tempo em testes manuais. Seria fundamental que esse analista pudesse ter seu foco totalmente na automação e manutenção da automação, para que futuramente, o tempo de automação de um teste também possa ser reduzido.

- Automatizar outros testes já identificados e priorizados pela equipe:

Essa recomendação baseia-se na ideia de que outros testes já foram listados para serem automatizado. Na seção 4.4.4 deste trabalho, é apresentado que a equipe de testes já realizou o trabalho de identificar e priorizar os cenários de testes para a automação, por tanto o próximo passo é aderir esses testes à automação.

- Estruturar a automação dentro do processo da equipe de testes:

O projeto de automação de testes iniciou-se como piloto, ou seja, ele não faz parte do processo da equipe de testes. Conforme abordado, a automação exige um investimento inicial grande, sendo necessário por tanto estruturar o processo de automação e enraizar e engajar a equipe sobre o tema.

- Definir e gerar as métricas de execução de testes:

Analisar os desafios em geral, proveitos, infortúnios e propósitos da equipe para então definir e gerar as métricas necessárias para alcançar os objetivos pressuposto.

- Acompanhar métricas e definir plano de ações através dos dados obtidos:

As métricas são necessárias para elaborar plano de ações de prevenções e melhorias dos processos envolvidos, bem como apresentar insumos aos gestores para tomadas de decisões.

5 CONCLUSÕES

Desde o início, este trabalho teve como intuito apresentar, através de um estudo de caso sobre como foi realizada a automação de testes da equipe do Solar BPM pertencente a empresa Softplan. O propósito era, além de estudar o que a literatura fala a respeito do assunto, comparar o modelo implantado na equipe com o que a literatura diz a respeito do tema e ao final, se necessário, gerar uma série de melhorias e recomendações para a empresa.

Para que todos os objetivos traçados fossem alcançados pelos autores, o início do projeto foi marcado por uma extensa pesquisa sobre automação de testes de software para ajudar a compor a introdução e a revisão bibliográfica do trabalho.

Como foi possível observar, o capítulo 4 tem início com a apresentação da empresa estudada e, de forma resumida, o processo de desenvolvimento dentro dela, bem como o processo dentro da equipe de testes, que foi determinante para que os autores pudessem se aprofundar o suficiente para trazer as informações mais relevantes.

A seguir foi realizada uma detalhada descrição do projeto piloto de automatização, com as ferramentas e tecnologias utilizadas como o Cucumber, linguagem Ruby, Capybara e Rspec.

A partir desse ponto, a equipe priorizou analisar importância e os desafios enfrentados ao iniciar um projeto de automatização de testes de software, o que possibilitou uma forma simples de apresentar o estudo, além de mostrar de forma clara e objetiva as a diferenças entre o processo de testes manual e automatizado além das vantagens que o processo de automatização pode trazer a uma empresa.

Verificou-se com esse estudo que o processo de automação pode reduzir e muito o tempo que um analista de testes empreende em uma tarefa quando comparado com o processo manual. Essa redução pode variar de acordo com inúmeras variáveis, como o tipo de teste a ser automatizado e a quantidade de pessoas disponíveis para a implementação dos testes, porém ainda assim, o estudo apontou que o trabalho na automação é recompensado com o tempo.

Outro ponto importante, levantado pelos autores, está relacionado aos custos iniciais que a Softplan teve. Por mais que as ferramentas utilizadas sejam *open source* (gratuitas), houve um investimento de “mão-de-obra" que implementou a automação e não trabalhou nos testes manuais por um período.

Portanto, fica evidenciado que todos os objetivos apontados pela equipe foram satisfeitos quando findado o trabalho com a apresentação da lista de recomendações e melhorias a serem apresentada pela empresa.

Foram listados como trabalhos futuros os seguintes tópicos:

- Automatizar os cenários de exceção dentro dos testes de regressão.
- Formalizar um padrão de código para automação de testes que atenda às necessidades da empresa.
- Analisar ferramentas proprietárias e *open source* utilizadas em automação de testes.
- Estudar como a implantação do BDD afeta o processo de desenvolvimento de software.
- Pesquisar o uso, geração e gestão de massas de dados para uso em testes automatizados.

REFERÊNCIAS

ABREU, Bruno. **4 grandes tendências de TI e os desafios para a área de QA**. 2019. Disponível em: https://blog.onedaytesting.com.br/tendencias-ti-desafios-qa. Acesso em: 24 mai. 2019.

ANICHE, Maurício. **TDD**: Test-Driven Development. [S. l.], 2 abr. 2014. Disponível em: https://tdd.caelum.com.br. Acesso em: 30 abr. 2020.

ARAÚJO, Matheus. **Por que automatizar testes?** 2018. Disponível em: http://dtidigital.com.br/blog/por-que-automatizar-testes. Acesso em: 18 mai. 2019.

ARCANJO, Brendo Felipe Rodrigues. Desenvolvimento orientado a comportamento (BDD) com Cucumber. [S. l.], In: **Devmedia**, 2015. Disponível em: https://www.devmedia.com.br/desenvolvimento-orientado-a-comportamento-bdd-com-cucumber/33547 Acesso em: 2 abr. 2020.

BARTIÉ, Alexandre. **Garantia da qualidade de software: As melhores práticas de Engenharia de Software aplicadas à sua empresa**. Rio de Janeiro: Campus, 2002.

BERNARDO, Paulo Cheque. **Padrões de Testes Automatizados**. 2011. 197 p. Dissertação (Mestrado em ciências da computação) - USP, São Paulo, 2011. Disponível em: https://teses.usp.br/teses/disponiveis/45/45134/tde-02042012-120707/publico/TestesAutomatizados_PauloCheque_Dissertacao.pdf. Acesso em: 30 abr. 2020.

CAETANO, Cristiano. **Automação de testes**. 2008. Disponível em: https://www.devmedia.com.br/automacao-de-testes/10249. Acesso em: 24 mai. 2019.

CAMPOS, Fabrício Ferrari de. **Melhorando os testes com o Rspec**. [S. l.], 14 mar. 2011. Disponível em: https://qualidadebr.wordpress.com/2011/03/14/melhorando-os-testes-com-o-rspec. Acesso em: 8 maio 2020.

CAPELAS, Bruno; ROMANI, Bruno; WOLF, Giovanna. IPhone 11: **Apple lança três novos celulares; câmera tripla é destaque: Eles custarão a partir de US$ 700; modelo mais caro sai por US$ 1.450**. O Estadão, São Paulo, 10 set. 2019. Disponível em: https://link.estadao.com.br/noticias/gadget,iphone-11-apple-lanca-nova-geracao-do-celular,70003004662. Acesso em: 10 set. 2019.

CARVALHO, André. **BDD: desenvolvimento orientado a comportamento.** [S. l.], 30 jul. 2019. Disponível em: https://inside.contabilizei.com.br/bdd-desenvolvimento-orientado-a-comportamento-62e71f2eabe9. Acesso em: 23 abr. 2020.

CONTROL, Prime. **BDD in 3 Minutes**. Disponível em: https://hackernoon.com/bdd-in-3-minutes-c3f8fc022237. Acesso em: 16 abr. 2020.

CUCUMBER. [S. l.]: **Cucumber Github.** [S. l.], 2020. Disponível em: https://github.com/cucumber/cucumber. Acesso em: 29 abr. 2020.

GIL, Antônio Carlos. **Como elaborar projetos de pesquisa.** 4. ed. São Paulo: Atlas, 2002.

HACKERNOON. **10 ferramentas de automação de testes mais usadas.** [S. l.], 18 maio 2018. Disponível em: https://www.primecontrol.com.br/10-ferramentas-de-automacao-de-testes-mais-usadas. Acesso em: 18 nov. 2019.

HUMBLE, Jez; FARLEY, David. Entrega contínua: como entregar software de forma rápida e confiável. 1. ed. Porto Alegre: Bookman, 2014.

HÜBNER, Patrícia Cristina. **Análise de Pontos de Função**: Uma técnica para medição de projetos de software sob o ponto de vista do usuário. [S. l.], 2020. Disponível em: http://micreiros.com/analise-de-pontos-de-funcao. Acesso em: 30 abr. 2020.

ISTQB, International Software Testing Qualifications Board**. Foundation Level Syllabus** – BSQTB, 2018.

LARA, Samyra. **Testes automatizados com Cabybara.** [S. l.], 29 mar. 2017. Disponível em: https://medium.com/jaguaribetech/testes-automatizados-com-cabybara-579e9688e3ab. Acesso em: 8 maio 2020.

MAHESHWARI, Shikha. **A Comparative Analysis of Different types of Models in Software Development Life Cycle. International Journal of Advanced Research in Computer Science and Software Engineering**. 2012. Disponível em: https://pdfs.semanticscholar.org/386c/031f69eda698e937997dcda17030448a4277.pdf. Acesso em: 05 abr. 2019.

MASTERTECH. **Como escolher uma linguagem de programação de aplicativos?** [S. l.], 9 out. 2018. Disponível em: https://usemobile.com.br/linguagens-de-programacao-de-aplicativos. Acesso em: 29 abr. 2020.

MATSUMOTO, Yukihiro. **Ruby in a Nutshell: A desktop quick reference.** 1. ed. [S. l.]: O'Reilly Media, Inc. 2001. 218 p. ISBN 9780596002145.

MILANI, Paula. **O FANTÁSTICO MUNDO DA CAPYBARA**: Usando cucumber e capybara para seus testes automatizados. [S. l.], 4 set. 2019. Disponível em: https://www.linkedin.com/pulse/o-fant%C3%A1stico-mundo-da-capybara-paula-milani. Acesso em: 8 maio 2020.

MOLINARI, Leonardo. **Testes Funcionais de Software**. Florianópolis: Visual Books, 2008. 214 p.

MOTTA, Alexandre de Medeiros; KINDERMANN, Conceição Aparecida; COLLAÇO, Gabriel Henrique; PRUDÊNCIO, Perpétua Guimarães; LEONEL, Vilson. **Universidade e ciência**: livro didático/conteudistas, Alexandre de Medeiros Motta, Conceição Aparecida Kindermann, Gabriel Henrique Collaço, Perpétua Guimarães Prudêncio, Vilson Leonel; design instrucional Eliete de Oliveira Costa. – Palhoça: UnisulVirtual, 2013. 158 p.: il.; 28 cm.

OLIVEIRA, Thais. **Automação de testes: o que é, quando e por que automatizar**. 2018. Disponível em: https://medium.com/venturus/quais-as-raz%C3%B5es-para-automa%C3%A7%C3%A3o-de-testes-c177cbd9397. Acesso em: 20 abr. 2019.

PORTAL GSTI, **O que é um teste de software?** Florianópolis, 2019. Disponível em: https://www.portalgsti.com.br/testes-de-software/sobre. Acesso em: 20 nov. 2019.

PRESSMAN, Roger**. Engenharia de software** – Uma abordagem profissional. Edição 8. AMGH Editora LTDA, 2016

RAIM, Samyr Abdo Nunes. **Os 5 grandes desafios no processo de Desenvolvimento de Software**. 2010. Disponível em: https://www.oficinadanet.com.br/artigo/desenvolvimento/os_desafios_no_processo_de_desenvolvimento_de_software. Acesso em: 05 abr. 2019.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software**. 3ª. ed. rev. e aum. Rio de Janeiro: Alta Books, 2013. 290 p.

ROCHA, Fabio Gomes. **TDD**: Fundamentos do Desenvolvimento Orientado a Testes. Aracajú -SE, 2013. Disponível em: https://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151. Acesso em: 5 maio 2020.

RUBY, Comunidade. **Ruby: O melhor amigo do programador.** [S. l.], 2020. Disponível em: https://www.ruby-lang.org/pt/about/. Acesso em: 29 abr. 2020.

SILVA, Edna Lúcia da. MENEZES, Estera Muszkat. **Metodologia da Pesquisa e Elaboração de Dissertação**. Florianópolis, 2005. Disponível em: Acesso em: 09 nov. 2019.

SOFTDESIGN. **Por que automatizar testes de software?** Disponível em: http://www.softdesign.com.br/blog/por-que-automatizar-testes-de-software. Acesso em: 18 maio 2019.

SOFTPLAN. **A Softplan**. Disponível em https://www.softplan.com.br/quem-somos Acesso em 05 dez. 2019.

SOFTPLAN. **SolarBPM**. Disponível em <https://www.solarbpm.com.br Acesso em 05 dez. 2019.

SOMMERVILLE, Ian**. Engenharia de software**. Edição 8. São Paulo: Pearson Addilson-Wesley, 2007 2012 1 recurso online ISBN 9788577808625.

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira; ALBERT, Renato Machado. **Analise de Pontos de Função**: Medições, estimativas e gerenciamento de projetos de software. 12. ed. rev. São Paulo: Ética Ltda, 2012. 230 p.