UNIVERSIDADE FEDERAL DE SANTA CATARINA

ESTUDO DE CASO DA UTILIZAÇÃO DE GHERKIN PARA AUTOMATIZAÇÃO DE TESTES E OBTENÇÃO DE MÉTRICA DE COBERTURA DE REQUISITOS

FLORIANÓPOLIS, SC

DEZEMBRO DE 2020

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

ESTUDO DE CASO DA UTILIZAÇÃO DE GHERKIN PARA AUTOMATIZAÇÃO DE TESTES E OBTENÇÃO DE MÉTRICA DE COBERTURA DE REQUISITOS

LARISSA TAW RUMIANA DE OLIVEIRA

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação.

FLORIANÓPOLIS, SC

DEZEMBRO, 2020

LARISSA TAW RUMIANA DE OLIVEIRA

ESTUDO DE CASO DA UTILIZAÇÃO DE GHERKIN PARA AUTOMATIZAÇÃO DE TESTES E OBTENÇÃO DE MÉTRICA DE COBERTURA DE REQUISITOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação.

_____

Prof. Orientador: Raul Sidnei Wazlawick

_____

Coorientadora: Bárbara Cabral

_____

Coordenador de projetos: Renato Cislaghi

_____

Membro da Banca Avaliadora: Gustavo Lobo

_____

Membro da Banca Avaliadora: Leonardo Souza

Ao pedacinho de terra

perdido no mar, que fez de

um ponto flutuante o seu lar.

AGRADECIMENTOS

Agradeço primeiro a Deus que sempre esteve presente em minha vida me dando esperança para continuar. Depois a minha mãe, por ensinar que no mundo estamos sozinhos e que somos o único responsável pelo próprio presente. Ao meu pai, que me proporcionou desde cedo o contato com a tecnologia. A minha avó que me fez ser resiliente e foi exemplo em princípios e valores. As minhas tias, que cada uma à sua maneira me incentivaram a estudar na Universidade Federal de Santa Catarina. Aos professores que não apenas repassaram conhecimento, mas foram guias para a vida. Agradeço também ao Laboratório Bridge, o qual estagiei por quase toda a graduação e que me orientaram não apenas na parte profissional, mas também na acadêmica e pessoal. Por fim, aos amigos que me acompanharam nessa jornada e contribuíram amparando igualmente a uma família e não sendo menos importantes dos que os citados anteriormente, pelo contrário, foram a base necessária para que eu conseguisse chegar até o fim da graduação e o sustento da fé de que esta é apenas a primeira etapa de um longo caminho a ser percorrido no encontro dos meus objetivos.

SUMÁRIO

RESUMO

O presente trabalho apresenta o estudo de caso da utilização da linguagem Gherkin como alternativa a documentação tradicionalmente utilizada pelo Laboratório Bridge. Introduz as dificuldades e necessidades que direcionaram à uma pesquisa de mercado para identificação do que poderia ser melhorado dentro do processo de qualidade de testes automatizados, como também justifica a decisão por optar-se por um projeto piloto para o uso de Gherkin. Além disso, descreve o porquê da busca pela metrificação da cobertura de requisitos e os resultados reais da introdução da linguagem Gherkin e da automatização com ela obtida por meio da criação de cenários de testes realizados com o Cucumber e Selenium, dentro de um escopo delimitado do sistema de controle de horário desenvolvido internamente.

Palavras-chave: G herkin - cucumber - teste automatizado - rastreabilidade

ABSTRACT

This paper presents the study case of the use of Gherkin language as an alternative to the documentation traditionally used by the Bridge Laboratory. It introduces the difficulties and needs that drive market research to identify what could be improved within the automated testing quality process, as well as justifying the decision to opt for a pilot project for the use of Gherkin. In addition, it describes why the pursuit of metering requirements coverage and the actual results of introducing and automating the Gherkin language by creating automated test scenarios with Cucumber and Selenium within a limited system scope internally developed time tracking system.

Key words: *Gherkin - cucumber - automated testing - traceability*

LISTA DE FIGURAS

LISTA DE REDUÇÕES

CTC	Centro Tecnológico
EFI	Espaço Físico Integrado
OKR	*Objectives and Key Results*
GT	Grupo de Trabalho
PDCA	*Plan, Do, Check, Act*

1 INTRODUÇÃO

O presente estudo de caso se iniciou devido a necessidade identificada no Laboratório Bridge, organização sem fins lucrativos integrada ao Centro Tecnológico (CTC) da Universidade Federal de Santa Catarina e que atua na pesquisa e desenvolvimento de soluções voltadas para a sociedade e financiadas pelo governo. O primeiro projeto desenvolvido foi o e-SUS AB, que começou em janeiro de 2013 no Espaço Físico Integrado (EFI) com disponibilidade para 30 pessoas, e continua em expansão, tendo mudado de endereço em outubro do mesmo ano para uma sala maior no prédio comercial “Max & Flora” devido ao aumento do número de colaboradores, que atualmente é composto por aproximadamente 120 pessoas.

Contudo, ter uma posição de sucesso e uma visão otimista não garantem que o crescimento se sustentará da mesma forma ao longo dos anos. Nesse contexto, no início de 2019 o laboratório buscou por melhorias no processo de gestão e começou a construção de um planejamento estratégico orientado pelo método OKR (*Objectives and Key Results*), finalizado em julho de 2019 pela equipe de gestão. Para que os objetivos do laboratório fossem alinhados, adotou-se a técnica de desdobramento denominada *top-down* e *bottom-up*, por meio do qual se busca que os objetivos do setor de posição mais alta do organograma esteja de acordo com o do setor mais baixo.

O principal objetivo do laboratório é a satisfação dos seus *stakeholders*, que é o GT (Grupo de Trabalho) do Ministério da Saúde, os colaboradores e a sociedade, por tanto a execução dos projetos no que diz respeito a qualidade é a prioridade, sendo o foco do planejamento estratégico. Diante disso, optou-se por

iniciar o planejamento por fases, selecionando primeiro a equipe de Automatizado para posteriormente realizar com as demais equipes de forma progressiva até toda a organização estar autogerenciável e alinhada. A escolha da equipe se deve primeiramente por estar diretamente ligada com a qualidade das entregas e consequentemente com a satisfação das partes interessadas citadas anteriormente. Depois, pelo interesse de um dos seus colaboradores por conhecimento de gestão em um momento oportuno, no caso a autora do presente estudo que atuou dentro da equipe do Automatizado quando o laboratório iniciava o seu planejamento estratégico.

Durante o primeiro semestre do ano de 2019, foram realizadas as capacitações dos colaboradores em metodologias de gerenciamento de equipes ágeis, como por exemplo o Kanban, que possibilita acompanhar o desempenho da equipe por meio do *Lead Time*, Coeficiente de Variação, Throughput e Taxa de serviço. Porém, esses dados não indicam diretamente informações em relação a qualidade, havendo a necessidade de métricas complementares.

A partir disso, realizou-se o benchmark com empresas de tecnologia que fabricam softwares, para verificar como elas estavam trabalhando e medindo a qualidade de seus produtos. As empresas foram escolhidas a partir da consultoria com a Roberta Lingnau de Oliveira, profissional com 14 anos de experiência na área de qualidade de software, que contribuiu com o presente trabalho sugerindo algumas métricas e ferramentas, além das indicações de outros profissionais relevantes à pesquisa.

Foram consultadas três empresas sobre métricas de qualidade, entre elas a ArcTouch, na qual se verificou que a mesma utilizava da linguagem Gherkin para a

criação de cenários e desenvolve uma “documentação viva” que possibilita a obtenção da cobertura de requisitos, informação de grande interesse pela área de qualidade, mas de difícil cálculo sem uma ferramenta de automação de métricas que considere a rastreabilidade como base. Uma vantagem dessa linguagem é a oportunidade de automatizar os cenários com o uso do Cucumber junto com o Selenium, sendo que a segunda ferramenta já é de conhecimento da equipe do Automatizado.

Nesse contexto, surgiu dentro do projeto de planejamento estratégico o interesse pelo Gherkin, porém com a necessidade de uma análise mais profunda, para avaliação de quais seriam as vantagens e desvantagens em adotá-la, considerando que a documentação atual é estável, com um grande número de requisitos e diversas regras de negócios complexas. Outro fator importante, trata-se da mudança no processo de trabalho que exigiria não apenas para o setor de qualidade, mas principalmente para os analistas do projeto, sendo que estes possuem uma rotina de trabalho de análise e documentação já enraizada na cultura do laboratório.

Para fins de decisões de mudanças de processo de trabalho, julgou-se necessário um estudo de caso para não descartar a aplicabilidade da linguagem no projeto como ferramenta alternativa ao modo atual de documentar os requisitos. Para tanto, optou-se por realizar o estudo em um escopo menor, selecionando o módulo principal do sistema Meu Bridge, plataforma desenvolvida internamente para controle de horário.

1.1 JUSTIFICATIVA

O presente trabalho descreve as dificuldades e necessidades que direcionaram ao estudo de caso da linguagem Gherkin e a busca pela obtenção da métrica de cobertura de requisitos, apresentando os resultados obtidos durante e após sua implementação. A principal finalidade é contribuir com o Laboratório Bridge e as demais instituições na decisão por adotar ou não essa alternativa de documentação, baseando-se em dados reais sobre seus desafios e benefícios.

1.2 OBJETIVOS

1.2.1 Objetivos Gerais

O objetivo geral deste trabalho é apresentar os resultados da utilização de uma documentação viva criada com cenários em linguagem Gherkin, como também de apresentar os impactos e resultados na automatização dos casos de teste e na obtenção da métrica de cobertura de requisitos por meio da rastreabilidade.

1.2.2 Objetivos Específicos

- Desenvolver cenários em Gherkin para o módulo de “Registro diário” do sistema Meu Bridge;
- Automatizar os cenários com o Cucumber e o Selenium;
- Medir a cobertura de requisitos para o módulo automatizado;
- Analisar as vantagens e desvantagens na adoção da prática.

1.3 METODOLOGIA

Inicialmente buscou-se adquirir os conhecimentos necessários para a implementação de um planejamento estratégico, como também o entendimento do processo de trabalho da equipe de gestão, do que estava sendo desenvolvido e das ferramentas que estavam sendo utilizadas. A capacitação envolveu desde conhecimentos básicos de gestão até os mais específicos, focando em assuntos relacionados com gestão ágil, planejamento estratégico e qualidade.

O período dedicado exclusivamente ao estudo descrito anteriormente, ocorreu por cerca de três meses, sendo o aprendizado concomitante ao desenvolvimento do planejamento e da execução do plano de ação definido para o mesmo. Deste modo, após o estudo teórico foi realizada uma pesquisa de campo, buscando identificar o estado atual da equipe Automatizado e quais as necessidades para alinhamento de suas atividades com os objetivos do laboratório.

A pesquisa empírica foi efetuada em primeiro plano pela autora do presente estudo, por meio da experiência de trabalho, atuando por 4 anos na área de qualidade, dentro dos quais 1,5 foram exclusivamente na equipe de Automatizado; em segundo, por meio de uma amostragem de duas equipes ágeis, entrevistando os *testers* e verificando como cada um trabalhava e tratava a parte de testes automatizados. O resultado das informações adquiridas foram validadas com o líder da equipe Qualidade, da equipe Automatizado e pela equipe Gestão, garantindo que as informações estavam devidamente coletadas e definidas.

Identificado o fluxo de trabalho atual dos testes automatizados dentro da equipe Automatizado e das equipes ágeis, foi realizado um *benchmark* em outras

organizações com pessoas com experiência na área de qualidade. O objetivo era identificar o que as outras organizações estavam fazendo para acompanhar e mensurar a qualidade de seus produtos ou serviços, como também a satisfação de seus clientes.

O tempo de maior dedicação do planejamento estratégico esteve na captação e desenvolvimento de conhecimento, pois é um ato que de acordo com o Falconi (2003) é imprescindível para reduzir as incertezas no processo de tomada de decisões. Os meios de adquiri-lo são principalmente através de recursos externos, com consultores, técnicos, professores, análise da literatura, visita a outras empresas, congressos, entre outros.

Concluída a etapa de absorção dos insumos necessários para a obtenção de *insights* e para uma análise mais apurada do estado atual da equipe Automatizado, foi identificada as métricas de interesse para o acompanhamento do desempenho, avaliado seus impactos e meios de extraí-los. A partir desses resultados, foram criados dois planos de ações para o alcance dos objetivos estabelecidos, sendo exposto neste trabalho apenas o qual está voltado para a utilização da linguagem Gherkin na automatização dos cenários de teste a para a obtenção da Cobertura de Requisitos.

Posteriormente, executou-se o plano de ação, sendo realizadas em paralelo pesquisas complementares relacionadas ao estudo tratado, abordando tópicos mais específicos ao tema e que estão expostos no capítulo de Fundamentação Teórica. Executou-se por fim a construção dos cenários com a linguagem Gherkin, a automatização dos mesmo por meio das ferramentas Selenium e Cucumber, a extração da métrica de Cobertura de Requisitos para o módulo do sistema

delimitado como escopo e a avaliação dos resultados, suas vantagens e desvantagens na utilização

1.5 ORGANIZAÇÃO DO TEXTO

O trabalho está organizado em cinco capítulos, sendo no primeiro abordado os objetivos gerais e específicos, assim como a metodologia utilizada para o desenvolvimento do mesmo. No segundo, está fundamentado os conceitos necessários para entendimento do tema abordado. Posteriormente segue o capítulo três, apresentando todas as etapas e especificações que levaram ao estudo da utilização do Gherkin como ferramenta para desenvolvimento da documentação. No quarto, estão os resultados obtidos com a introdução de uma documentação escrita na linguagem Gherkin e da automatização dos cenários de teste. Por fim, no último capítulo está a conclusão em relação ao trabalho realizado.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo visa tratar da fundamentação teórica necessária para um melhor entendimento e desenvolvimento do presente trabalho. São abordadas as principais definições e características relacionadas ao tema tratado, começando pelos níveis de teste no intuito de esclarecer as diferenças entre os diferentes tipos. Posteriormente, uma breve introdução a metodologia de *Agile Testing* e os artefatos que serão utilizados para o desenvolvimento do estudo de caso (*User Storie*s e cenários em Gherkin). Por fim, as métricas na área de qualidade dando enfoque a Cobertura de Requisitos e um breve relato sobre o sistema para o qual será aplicado o estudo de caso.

2.1 NÍVEIS DE TESTE

A qualidade de um software pode ser avaliada em diferentes aspectos, sendo definida e comparada pelo ISTQB (2008) em diferentes níveis de teste de acordo com as especificações abaixo:

2.1.1 Teste de Componente/ Teste Unitário

O teste de componente ou teste unitário, podem ser conhecidos também como teste de módulo ou teste de programa e possuem como principal objetivo identificar defeitos nas funcionalidades dos softwares, verificando isoladamente cada componente, ou seja, os objetos, classes, etc.

2.1.2 Teste de Integração

Teste de integração busca verificar as interações entre diferentes componentes do sistema, avaliando como o sistema está funcionando em relação ao sistema operacional, ao sistema de arquivos e outros sistemas que possa ter alguma troca de informação.

2.1.3 Teste de Sistema

Teste de sistema verifica o comportamento do sistema, se está de acordo com o escopo definido e proposto antes do desenvolvimento, sendo realizado geralmente ao final do projeto para avaliar se atende as especificações definidas para o produto e tendo como objetivo encontrar o maior número de defeitos possíveis.

2.1.4 Teste de Aceitação

O teste de aceitação é na maioria das vezes de responsabilidade do usuário ou cliente final, apesar de envolver também outras partes interessadas. O objetivo principal é validar o sistema e estabelecer uma certa confiança.

2.1.5 Teste Funcional

O teste funcional é baseado nas funções do sistema, podendo ser realizado em qualquer nível de teste. São geralmente especificados em uma documentação e podem ser feitos com bases nos requisitos ou processos do negócio.

2.1.6. Teste de características do produto

Conhecido também como testes não-funcionais, pois buscam validar o quão rápido ou bem feito o sistema está executando determinada funcionalidade, sendo a validação do funcionamento tratado em outros níveis de testes descritos anteriormente. A seguir estão os tipos de testes que avaliam as características do produto:

2.1.6.1 Teste de Confiabilidade

Avalia o sistema em relação a tolerância a falhas, a capacidade de recuperação de um problema e a conformidade com o esperado.

2.1.6.2 Teste de Usabilidade

Avalia o quão fácil é um sistema em relação ao seu uso e a promoção da capacidade de aprender do usuário, como também a sua atratividade.

2.1.6.3 Teste de Eficiência

Avalia o comportamento do sistema, o quanto de recurso utiliza para desempenhar suas atividades.

2.1.6.4 Teste de Manutenibilidade

Avalia se o sistema é passível de ser analisado, alterado, testável, além de verificar a estabilidade e conformidade.

2.1.6.5 Teste de Portabilidade

Avalia o sistema em relação a adaptabilidade, instabilidade, coexistência, substituibilidade e conformidade.

2.1.7 Teste de Arquitetura

O teste estrutural é freqüentemente chamado de 'caixa branca' porque está interessado no que está acontecendo 'dentro da caixa', ou seja, no código. Pode ocorrer em qualquer nível de teste, embora tende a ser aplicado principalmente no teste de componente e de integração.

2.1.8 Teste de Regressão

O objetivo do teste de regressão é verificar se as modificações no software ou ambiente não causaram efeitos colaterais e que o sistema ainda atende aos seus requisitos, revisando testes que já foram executados anteriormente.

2.2 *AGILE TESTING*

Lisa Crispin e Janet Gregory (2009) define *Agile Testing* como um conjunto de atividades que giram em torno da escrita do código de produção dos testes e do desenvolvimento, de modo que todos da equipe são considerados desenvolvedores e devem estar focados em entregar um produto de qualidade.

Dentre as várias práticas básicas da metodologia ágil, estão a utilização de *User Stories* e Cenários, que são técnicas que utilizam de uma escrita em linguagem passível de execução e automação, gerando assim uma documentação

viva, na qual qualquer alteração nos requisitos impacte diretamente nos casos de testes e em seus resultados, criando a rastreabilidade entre o que este está sendo testado e o que que está desenvolvido.

2.2.1 *User Stories*

Conforme Cezerino e Nascimento (2016) os *User Stories* pertencem as metodologias ágeis de desenvolvimento, ajudando os *stakeholders* a partilhar os requisitos que devem ser implementados, deixando claro o porquê são necessários e quais valores de negócio apresentam.

Segundo ainda os autores, o *User Story* deve ser composto por pequenos cenários escritos em uma linguagem próxima do entendimento das partes interessadas, para gerar uma comunicação eficaz entre todos, gerando discussões e novas conversas que guiem a equipe na implementação.

Outros pontos importantes são as propriedades que cada *User Story* deve ter e que estão listadas a seguir:

1. Descritiva: deve ser possível extrair uma expectativa ou funcionalidade;
2. Estimável: deve ser possível estimar o esforço para a sua realização;
3. Testável: deve ser possível testar e verificar se está de acordo as expectativas dos *stakeholders*.
4. Categorizável: deve ser possível categorizar em grau de importância, priorizando de acordo com a necessidade.

Mike Cohn (2009) diz que o *User Story* não deve abranger muitos detalhes das funcionalidades, pois pode ocasionar desperdício de tempo e retrabalho

considerando que é comum a mudança nos requisitos durante o desenvolvimento, sendo melhor restringir em *"role, function, business value"*, ou seja, na função que determinado papel deve realizar de acordo com um objetivo determinado.

2.2.1 *Gherkin*

Gherkin é uma linguagem executável, próxima a linguagem natural e que utiliza de palavras chaves para descrever processos de acordo com uma estrutura interpretável.

2.4 MÉTRICAS

Métricas servem para acompanhar o desempenho de uma equipe, com o objetivo de identificar possíveis problemas no processo de desenvolvimento das atividades ou para certificar-se de que estão sendo atingidos os resultados esperados.

> "Metrificar é o primeiro passo para o controle e eventualmente para a melhoria. Se você não consegue medir algo, você não consegue entendê-lo. Se você não consegue capturá-lo, você não consegue controlá-lo. Se você não consegue controlá-lo, você não consegue melhorá-lo. (A, Raphael Donaire; 2000, p. 8)"

Dentre as métricas de acompanhamento mais comum, estão as que são extraídas por meio da aplicação do método Kanban, que são o *Lead Time*, o

Coeficiente de Variação, o *Throughput* e a Taxa de Serviço. Em relação às mais específicas em relação a qualidade do produto, existe a de Cobertura de Requisitos, que exige a existência da rastreabilidade dos requisitos para ser medida.

2.4.1 Cobertura de Requisitos

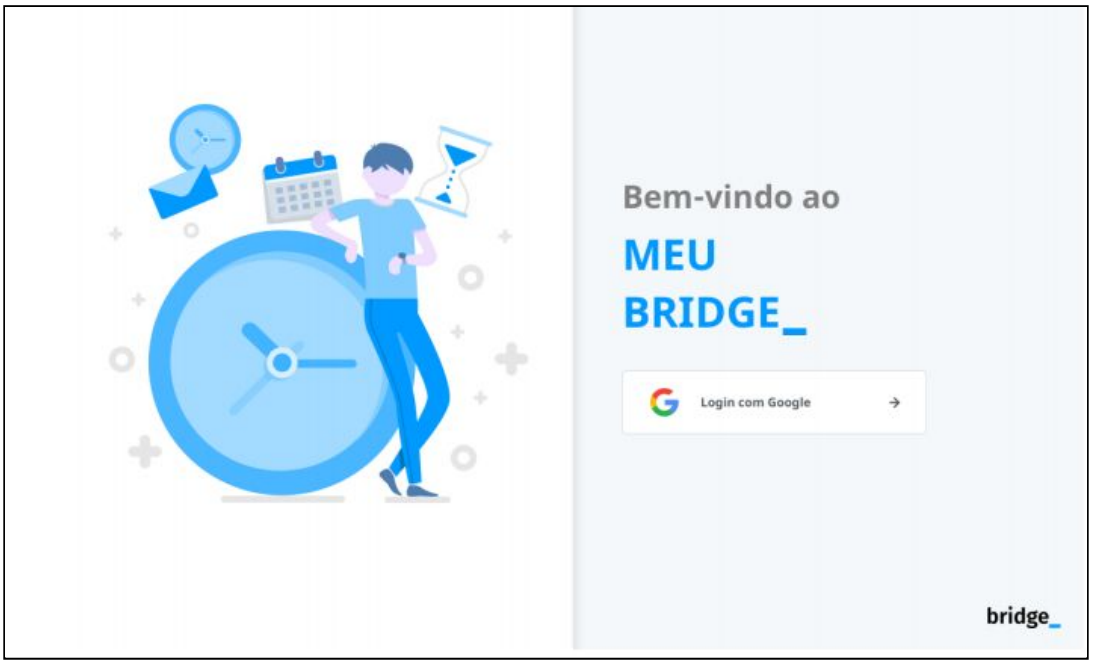
De acordo com Sayão e do Prado Leite (2005) a Cobertura de Requisitos analisa a rastreabilidade entre requisitos e casos de testes, identificando quais não foram previstos e necessitam de testes complementares. Porém, no caso do presente estudo, está métrica avalia quais testes dos previstos na documentação não foram automatizados e estão pendentes de desenvolvimento.

2.5 SISTEMA MEU BRIDGE

O sistema selecionado para o estudo de caso do presente trabalho foi o Meu Bridge ("meu.bridge.ufsc.br"), ferramenta web open source desenvolvida pelo Laboratório Bridge com o objetivo de auxiliar no acompanhamento da carga de horária de trabalho dos colaboradores, das ausências, atestados, férias, entre outras atividades em relação ao controle de horas.

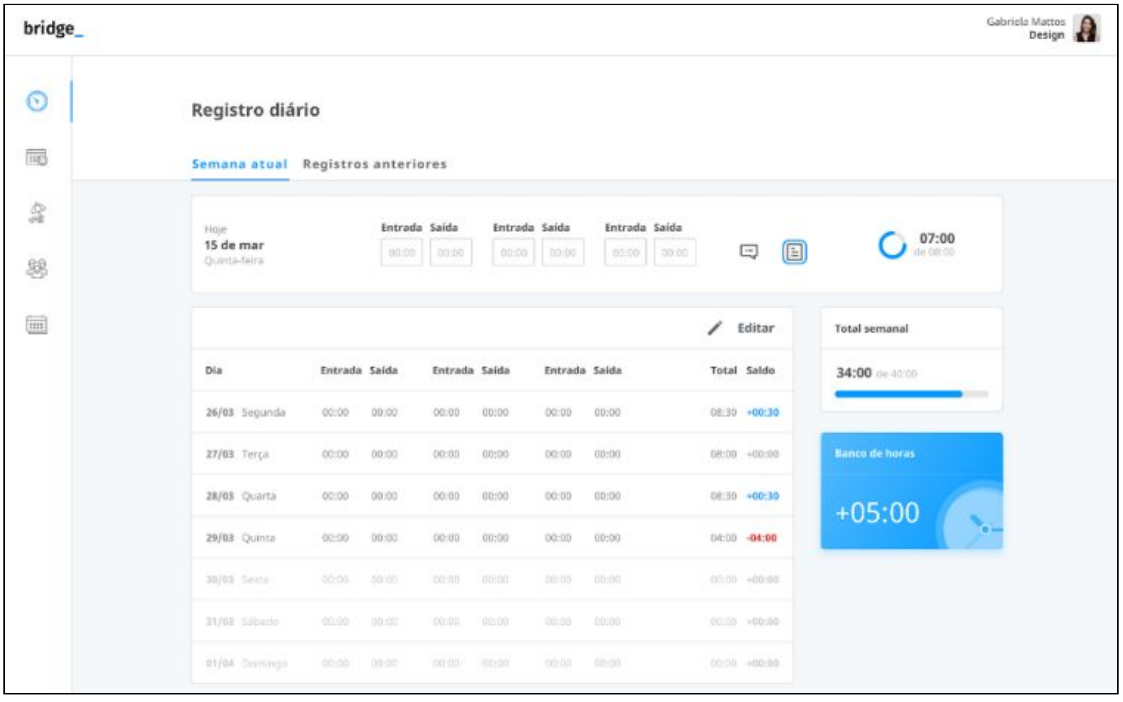
Abaixo está a imagem da tela de Login do sistema, seguida da imagem do módulo de Registro Diário, sendo estas as páginas delimitadas como o escopo do trabalho e que terão parte de suas funcionalidades aplicadas ao desenvolvimento do estudo da utilização de Gherkin para automatização dos testes e a obtenção da métrica de Cobertura de Requisitos.

Figura 1 - Tela de login do sistema Meu Bridge



Fonte: FRAGA e MOLINARI, 2018.

Figura 2 - Tela de controle de horas do sistema Meu Bridge



Fonte: FRAGA e MOLINARI, 2018.

3 PLANEJAMENTO ESTRATÉGICO

Planejar é estruturar o caminho antes da ação, definindo os objetivos de acordo com o posicionamento que se quer atingir. Para tanto, deve-se antes de tudo conhecer o estado atual, pois não é possível escolher um trajeto sabendo apenas o destino, é preciso saber o ponto de partida para decidir qual será a melhor rota e as ferramentas necessárias. Estratégia é a tomada de decisões diante das alternativas existentes, avaliando os impactos com base em fatos que apontem melhores resultados frente aos objetivos estabelecidos.

A falta de planejamento pode acarretar em mudanças constantes na direção dos esforços e consequentemente gerar um desperdício de recursos, além de impactar a motivação. O conhecimento do porquê se está fazendo algo, o acompanhamento de progresso e metas claras trazem a sensação de realização, contribuindo no engajamento dos colaboradores. Outro ponto relevante é a melhora na comunicação entre as equipes, sendo de extrema importância considerando que está diretamente ligada a maioria dos problemas de qualquer organização.

Antes de tudo, deve-se saber o estado atual para posteriormente realizar um planejamento de acordo com os objetivos que se quer atingir e então executá-lo. Para este processo, foi aplicada a metodologia A3 que foi desenvolvida com base no modelo PDCA (*Plan, Do, Check, Act*), o qual não terá maiores definições no presente trabalho por não se tratar diretamente do tema abordado, porém os seus principais resultados estão descritos nos tópicos a seguir:

3.1 DIAGNÓSTICO

Este capítulo apresenta os detalhamentos das partes envolvidas na construção do planejamento estratégico, desde a etapa de Acesso à Realidade, na qual ocorreram algumas análises e investigações dos processos de trabalho e dos resultados atuais, até a construção e execução do Plano de Ação após a definição do Estado Futuro desejável.

3.1.1 Acesso a Realidade

A realidade por vezes é diferente do que está visualmente aparente, fato que foi possível de constatar ao verificar que os *testers* da equipe Automatizado tinham uma visão diferente do fluxo de trabalho em relação ao *testers* que automatizam nas equipes ágeis, como também de seus líderes. Para entender a situação foi necessário ir ao *Gemba,* que de acordo com John Shook (2016), significa “lugar real” e descreve onde acontece realmente o trabalho. Estar atuando dentro da equipe Automatizado criava a falsa sensação de já estar no ambiente certo para acessar a realidade, porém muitas informações novas foram adquiridas após questionar como cada equipe atuava.

3.1.1.1 *Contexto*

A equipe Automatizado existe desde 2014, tendo gerado 8.463 testes automatizados para mais de 30 módulos do e-SUS AB. No mesmo ano do desenvolvimento do presente estudo (2019), o sistema estava passando por uma refatoração desenvolvida pela equipe REDESIGN, que foi criada com o objetivo de

atualizar o sistema para que o mesmo acompanhe as novas tecnologias e tendências do mercado, o que poderia afetar diretamente os resultados dos testes existentes.

Além disso, no início apenas a equipe Automatizado desenvolvia testes automatizados, tendo como base a documentação do sistema, realizando os testes do tipo caixa preta, ou seja, sem conhecimento do código. Porém, este cenário também passou por algumas alterações e tende a continuar mudando, pois agora os *testers* das equipes ágeis também criam testes automatizados, sendo a maioria teste de caixa-branca ou caixa-cinza.

3.1.2 Situação Atual

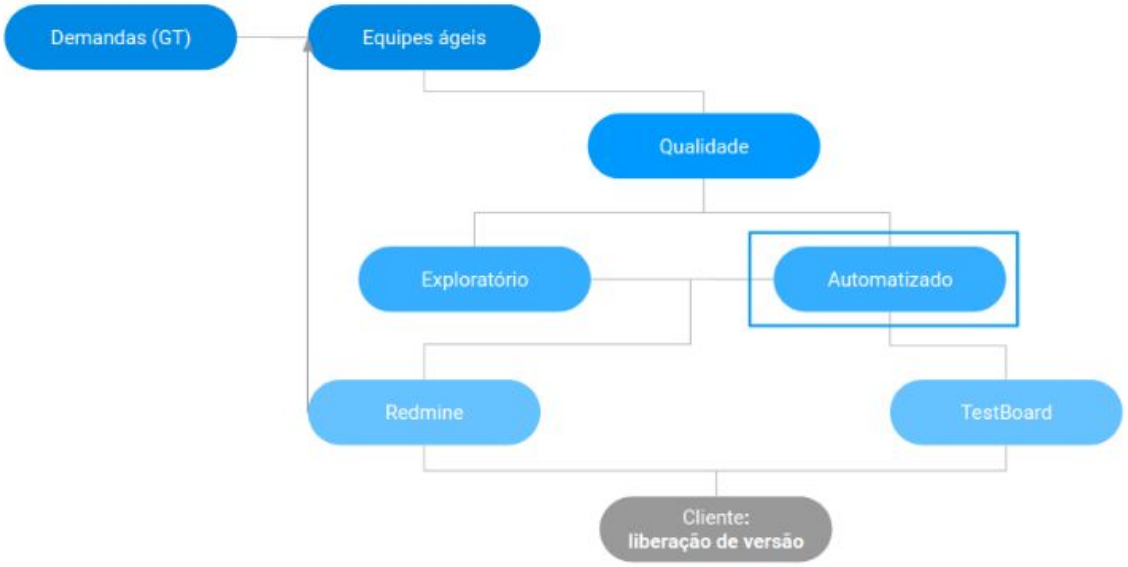
Os testes automatizados são adicionados ao Testboard para execução e agrupados por módulo, *build* e versão. Algumas equipes ágeis antes do desenvolvimento dos testes automatizados, realizam a criação de um documento com todos os possíveis casos de teste, enquanto que outras pensam apenas no cenário durante o desenvolvimento.

O Testboard é uma ferramenta criada pela equipe do Automatizado, tendo como papel centralizar os registros de execuções dos testes pertencentes a aplicação PEC. Foi implementada em 2015 e possui atualmente quase duas milhões de execuções armazenadas e realizadas para mais de 100 *releases*. A cada nova *build* lançada, um identificador é gerado para cada teste automatizado e enviado ao Testboard. Os detalhes sobre como são estruturados esses identificadores dentro da aplicação não serão abordados, porém algumas características são de relevância:

- Um teste pertence a uma *release* (versão + *build*) e a um módulo específico.
- Um teste pode possuir um dos quatro estados: Aprovado, Problema conhecido, Não executado, Inconsistente.

Para maior esclarecimento da estrutura organizacional do laboratório, segue abaixo a imagem da hierarquia do processo de trabalho em relação a área de qualidade. O *input* das atividades ocorre de acordo com as demandas do GT, sendo desenvolvidas e válidas em um primeiro momento pelas equipes ágeis, passando depois para a área de qualidade, que por sua vez é dividida entre testes exploratórios e testes automatizados. Toda tarefa de desenvolvimento é registrada no Redmine, assim como os *reports* de *bugs,* independentemente de por qual equipe foi feita a identificação. No caso dos testes automatizados, os *reports* do Redmine são associados aos testes executados no Testboard.

Figura 3 - Hierarquia do processo de trabalho da área de qualidade.



Fonte: elaborado por Larissa Taw R. de Oliveira, 2019.

O Redmine é uma ferramenta de *software* livre utilizada por colaboradores de diferentes áreas para o gerenciamento de tarefas, que representam desde novas funcionalidades a serem desenvolvidas até às solicitações de manutenção corretiva registradas a partir de *bugs* encontrados no sistema, o qual são denominadas como *reports*.

Inicialmente a equipe Automatizado era a única responsável pelo desenvolvimento dos testes automatizados, porém essa realidade mudou com a disseminação desse conhecimento para as equipes ágeis que passaram atuar com o papel principal nesta atividade. O Automatizado estava encarregado por executar e realizar a manutenção dos testes, como também da aprovação dos *pull requests* da própria equipe e das demais, verificando se os testes desenvolvidos estão construídos corretamente.

Figura 4 - Fluxo de trabalho geral atual

Fonte: elaborado por Larissa Taw R. de Oliveira, 2019.

No fluxo acima, o *input* 1 representa a entrada de demandas de testes automatizados para as equipes ágeis, que são geradas a partir do desenvolvimento de novas funcionalidades, melhorias, correções ou alterações em algum dos requisitos existentes. Conforme o tipo da demanda inicial a equipe ágil que desenvolveu a solicitação opta por realizar o teste de integração ou de sistema, sendo que para a segunda opção algumas desenvolvem os casos de teste com base na documentação antes de construir o teste de fato.

O *input* 2 trata das demandas do Automatizado, que são as manutenções dos testes que geralmente passam a apresentar o estado de inconsistente após a refatoração de algum módulo. Identificando que os testes não estão mais de acordo com os requisitos do sistema a equipe realiza a análise da documentação e compara com os cenários dos testes, efetuando as alterações necessárias para a atualização dos mesmos.

Os testes desenvolvidos ou atualizados pelas equipes ágeis são revisados pela pessoa mais experiente da equipe Automatizado, sendo aprovado e integrado com os demais testes disponíveis no Testboard caso estejam corretos. Posteriormente são executados e os resultados são analisados, de modo que todos devem estar definidos como “Aprovados” ou “Problemas conhecidos”, sendo a segunda situação para os casos de erros já relatados.

Na Figura 2, a atividade de “Desenvolvimento de casos de teste” e de “Análise dos testes e da documentação”, estão destacados pelos retângulos vermelhos, pois são atividades consideradas como retrabalho devido a falta de rastreabilidade da documentação. Caso os requisitos do sistema fossem documentados com cenários em Gherkin por exemplo, seria possível saber

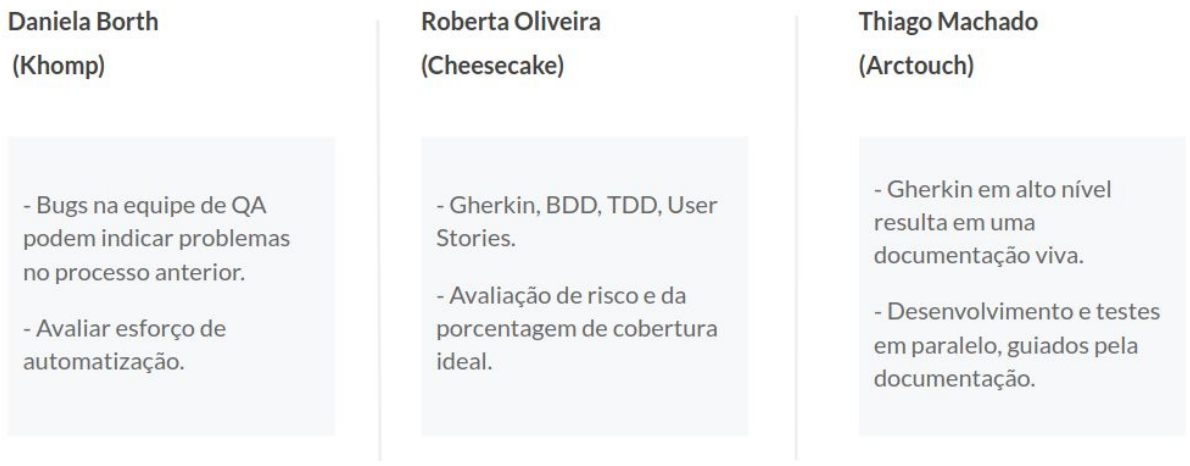
exatamente qual teste deve ser atualizado a partir da alteração do requisito, não havendo também a necessidade de construir casos de teste do zero, pois os próprios cenários podem ser automatizados por meio do Cucumber.

3.1.3 Benchmark

A pesquisa de *benchmark* possibilitou identificar novas maneiras de atuação na busca da garantia da qualidade, e quais os KPIs e as ferramentas utilizadas para a gestão do fluxo de trabalho e acompanhamento do desempenho estava sendo utilizados pelo mercado.

Os profissionais abaixo estão destacados com os pontos levantados por cada um e que foram considerados de maior relevância para o contexto do laboratório. Os repasses feitos foram com base no que julgaram importante ou útil para a gestão da qualidade, fazendo referências a ferramentas conhecidas, que não necessariamente estavam sendo utilizadas na empresa em que atuavam no momento.

Figura 5 - Benchmark



Fonte: elaborado por Larissa Taw R. de Oliveira, 2019.

3.1.5 Métricas de Acompanhamento

A equipe Automatizado é experiente, porém estava subjetivo o quão satisfatório e impactante estavam as atividades em relação a qualidade do produto e se realmente estaria entregando bons resultados, pois não há uma análise e comparação do histórico de trabalho baseado em métricas específicas.

Como exemplo, considere o cenário de uma viagem, para realizar a mesma deve-se planejar com base no ponto de partida e de destino, para posteriormente definir o caminho que será percorrido, podendo ser uma das métricas de desempenho manter a velocidade média em 60km/horas e percorrer no mínimo 500 km/dia. Podemos inferir que um motorista experiente saberia dizer o quanto ele percorrerá por dia e em qual velocidade, porém imprevistos ocorrem, como acidentes no trânsito, pneu furado, adoecimento do próprio motorista, sendo estas circunstâncias fora do controle e que podem atrasá-lo por algumas horas ou dias. Acompanhar esses incidentes, suas frequências, impactos e os meios utilizados para contornar a situação no intuito de cumprir com a meta estabelecida, geram históricos que podem ser utilizados para um melhor planejamento.

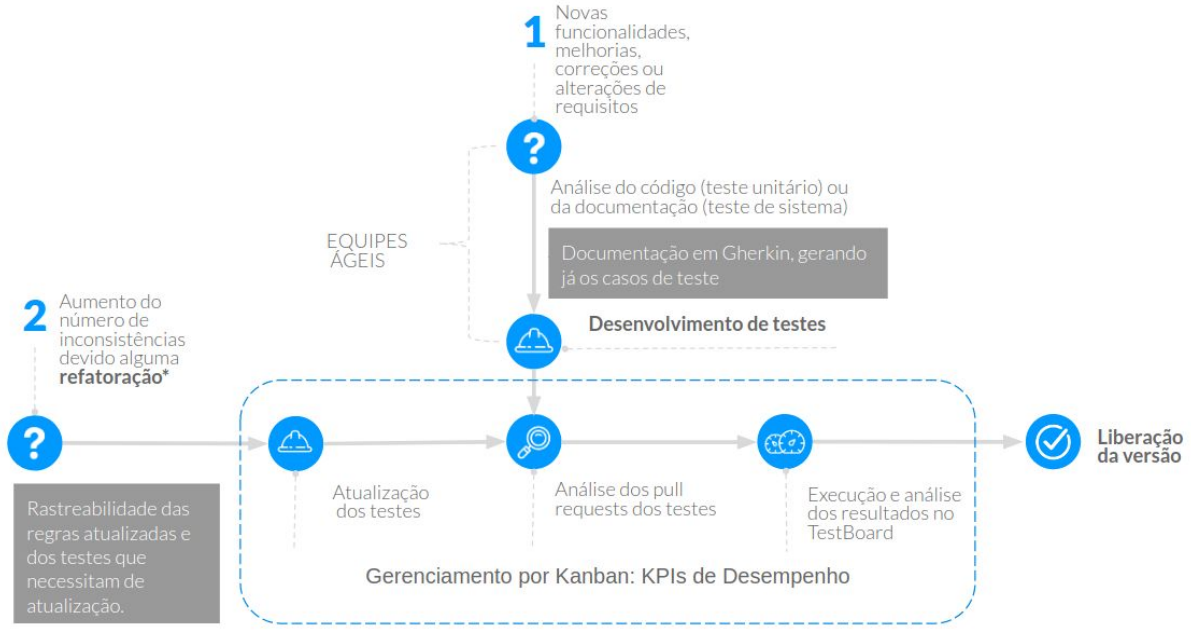
Seguindo essa linha de pensamento, caso observe-se que determinada via entre o ponto de origem e destino apresenta um número maior de acidentes do que outra que apesar de um pouco mais distante é mais segura, poderia ser feita uma análise de custos e benefícios e optar pela segunda alternativa de trajeto. No mesmo contexto, por meio de métricas em relação ao tempo das entregas, a evolução das atividades e dos resultados apresentados pela equipe, sendo possível fazer ajustes no processo de trabalho que possam melhor a jornada, através da

implementação de uma rotina de acompanhamento que permita a identificação prévia de possíveis problemas.

3.2 ESTADO FUTURO

Após a análise do estado atual, das métricas de interesse e de quais processos deveriam ser monitorados ou modificados, foi estabelecido o estado futuro, conforme o fluxo de trabalho apresentado na Figura 3.

Figura 6 - Fluxo de trabalho futuro



Fonte: elaborado por Larissa Taw R. de Oliveira, 2019.

Os processos que na Figura 2 estavam destacado em vermelho representam processos de retrabalho, estando agora substituídos pelas caixas de texto em cinza pelas novas atividades que podem vir a solucionar os problemas apontados anteriormente. Estão circulados em azul, os processos dos quais é possível se obter as métricas do Kanban para o acompanhamento do desempenho da equipe. A partir

das propostas para o estado futuro, foram criados dois planos de ação, o primeiro para implementar o Kanban, o segundo sendo o presente trabalho.

3.3 PLANO DE AÇÃO

O plano de ação para o estudo de caso da utilização de Gherkin para automatização de testes e obtenção de métrica de Cobertura de Requisitos foi delimitado para o módulo “Registro diário” do sistema “Meu bridge”. Abaixo segue as especificações do plano:

- Capacitação: tempo estimado em 40 horas-homem
    - Estudo sobre a utilização de *User Stories;*
    - Estudo sobre a utilização e desenvolvimento dos cenários em Gherkin;
    - Estudo da utilização do Cucumber e para automatização dos cenários.
- Implementação e execução : tempo estimado em 80 horas-homem
    - Criação de um novo modelo de documentação para o módulo definido para o estudo de caso, utilizando *user stories* e Gherkin;
    - Criação dos testes automatizados a partir dos cenários em Gherkin, utilizando o Cucumber como ferramenta;
    - Execução dos testes automatizados;
- Análise de Resultados: tempo estimado em 20 horas-homem
    - Avaliação das vantagens e desvantagens;
    - Obtenção da métrica de Cobertura de Requisitos;
    - Considerações finais sobre a aplicabilidade de um novo modelo de documentação para o contexto do laboratório Bridge.

4 IMPLEMENTAÇÃO E EXECUÇÃO

O processo da automatização dos casos de testes de uma documentação viva começam com a definição das *features*, ou seja, a descrição dos requisitos e funcionalidades para um determinado escopo, utilizando de uma linguagem passível de execução.

4.1 MODELO DE DOCUMENTAÇÃO

O modelo de documentação proposto é o composto por *User Stories* e cenários escritos em Gherkin. O *framework* escolhido e utilizado para a escrita dos cenários foi o Cucumber, que possui como linguagem original o inglês, porém também pode ser utilizado em outras línguas, como o português, desde que seja inserido no topo da *feature* a *tag* #language:pt.

A história do usuário deve ser escrita de acordo com a estrutura da figura abaixo e referenciada no arquivo de *feature* pela chave “Funcionalidade”.

Figura 7 - Estrutura de User Story em Português

```
Como um  [usuário]
Eu gostaria de [realizar uma ação]
A fim de acessar/ atingir [um objetivo específico]
```

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

Os cenários também possuem uma estrutura específica que deve ser referenciado no arquivo *feature* pela chave “Cenário”:

Figura 8 - Exemplo de Cenário em Português

Dado [um contexto]

Quando [ocorrer determinado evento]

Então [um resultado é esperado]

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

Para o estudo de caso foi definido como escopo o módulo do Registro Diário, porém para acessá-lo é necessário fazer o login no sistema Meu Bridge, tendo sido o primeiro exemplo a ser desenvolvido o arquivo login.feature:

Figura 9 - Código do arquivo login.feature

```
#language:pt
@login
Funcionalidade: Login no Meu Bridge
  Como um usuario normal
  Eu gostaria de ir para o meu bridge
  A fim de acessar a pagina principal

@failed @demo
Cenário: Login no Meu Bridge - visualizar Registro Diario
  Dado que eu acesso a pagina principal
  Quando eu realizo o login
  Então eu devo ver a pagina do registro diario

@failed @demo
Cenário: Login - Incluir Observação
  Dado que eu acesso a pagina principal
  Quando eu realizo o login
  E eu clico em Observação
  E eu digito a observação "Horas Abonadas"
  E eu clico em Salvar
  Então eu devo ver a observação salva com "Horas Abonadas"

@ready @demo
Cenário: Login - Incluir Observação
  Dado que eu acesso a pagina principal
  Quando eu realizo o login
  E eu clico em Dark mode
```

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

4.2 AUTOMATIZAÇÃO DOS CENÁRIOS

A automatização dos cenários exige que a documentação esteja

desenvolvida em uma linguagem executável, conforme o modelo apresentado no tópico anterior, para posteriormente criar os métodos dos passos dos cenários.

Figura 10 - Código da classe *LoginStepsDefinitions*.java

```
11  public class LoginStepsDefinitions {
12
13      @Steps
14      LoginUISteps loginUISteps;
15
16      @Step
17      @Screenshots(afterEachStep=true)
18      @Dado("^que eu acesso a pagina principal$")
19      public void que_eu_acesso_a_pagina_principal() {
20          loginUISteps.visit();
21      }
22
23      @Quando("^eu realizo o login$")
24      public void eu_realizo_o_login() {
25          loginUISteps.loginComGoogle();
26      }
27
28      @Então("^eu devo ver a pagina do registro diario$")
29      public void eu_devo_ver_a_pagina_do_registro_diario() {
30          loginUISteps.assertThatIsRegistroDiarioPage();
31      }
32
33      @Quando("^eu clico em Observação$")
34      public void eu_clico_em_Observacao() {
35          loginUISteps.clickOnObservacao();
36      }
37
38      @Quando("^eu digito a observação \"([^\"]*)\"$")
39      public void eu_digito_a_observação(String obs) {
40          loginUISteps.fillObservacao(obs);
41      }
42
43      @Quando("^eu clieo em Salvar$")
44      public void eu_clieo_em_Salvar() {
45          loginUISteps.clickOnSave();
46      }
```

Fonte: Elaborado por Larissa Taw R. de Oliveira,, 2019.

A figura acima da Classe *LoginStepsDefinitions*.java exibe a implementação dos passos que foram definidos no arquivo *login.feature* e faz a junção das informações da documentação com o código java que fará a execução do teste. Além dessa classe, foi criada a *LoginUISteps*.java, onde são instanciadas as

páginas que são acessadas para a interação com os elementos da tela do teste:

Figura 11 - Código da classe LoginUISteps.java

```
14  public class LoginUISteps extends ScenarioSteps {
15
16      @Page
17      LoginPage loginPage;
18
19      @Page
20      RegistroDiarioPage registroDiarioPage;
21
22
23      @Step
24      public void visit() {
25          loginPage.open();
26      }
27
28      @Step
29      public void loginComGoogle() {
30          loginPage.loginComGoogle();
31          loginPage.switchToPage(RegistroDiarioPage.class);
32      }
33
34      @Step
35      public void assertThatIsRegistroDiarioPage() {
36          registroDiarioPage.waitForPageLoaded();
37          registroDiarioPage.assertThatIsRegistroDiarioPage();
38      }
39
40      @Step
41      public void reallizar_login() {
42          visit();
43          loginComGoogle();
44      }
45
46      @Step
47      public void clickOnObservacao() {
48          registroDiarioPage.clickOnObservacao();
49      }
```

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

Cada página utiliza no teste é criada conforme o padrão de projetos denominado *PageObjects*, para a representação dos elementos que serão utilizados

para o teste. Neste caso, foi criado uma página chamada *LoginPage*.java e outra classe com o nome RegistroDiarioPage.java, para representar respectivamente a página de *Login* e de Registro Diário.

Figura 12 - Código do arquivo login.page

```
12  @DefaultUrl("page:login.page")
13  public class LoginPage extends PageObject {
14
15      @FindBy(id = "seta")
16      WebElementFacade loginGmail;
17
18      @FindBy(name = "identifier")
19      WebElementFacade username;
20
21      @FindBy(id  = "identifierNext")
22      WebElementFacade usernameNext;
23
24      @FindBy(name = "password")
25      WebElementFacade password;
26
27      @FindBy(id  = "passwordNext")
28      WebElementFacade passwordNext;
29
30      public void loginComGoogle() {
31          loginGmail.withTimeoutOf(20, TimeUnit.SECONDS).waitUntilVisible();
32          loginGmail.click();
33
34          username.withTimeoutOf(20, TimeUnit.SECONDS).waitUntilVisible();
35          username.sendKeys("larissataw@bridge.ufsc.br");
36          usernameNext.click();
37
38          password.withTimeoutOf(20, TimeUnit.SECONDS).waitUntilVisible();
39          password.sendKeys("17222015");
40          passwordNext.click();
41      }
42
43  }
```

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

Figura 13 - Código parcial do arquivo registrodiario.page

```
15  @DefaultUrl("page:registrodiario.page")
16  public class RegistroDiarioPage extends PageObject {
17
18      String url = "https://meu.bridge.ufsc.br/dailyregister/current/visualize";
19
20      @FindBy(tagName = "title")
21      WebElementFacade title;
22
23      @FindBy(className = "css-1ohfo2x")
24      WebElementFacade cheguei_button;
25
26      //Grupo 1
27      @FindBy(name = "activityIntervals[0].startTime")
28      WebElementFacade entrada_field_0;
29
30      @FindBy(name = "activityIntervals[0].endTime")
31      WebElementFacade saida_field_0;
32
33      //Grupo 2
34      @FindBy(name = "activityIntervals[1].startTime")
35      WebElementFacade entrada_field_1;
36
37      @FindBy(name = "activityIntervals[1].endTime")
38      WebElementFacade saida_field_1;
39
40      //Grupo 3
41      @FindBy(name = "activityIntervals[2].startTime")
42      WebElementFacade entrada_field_2;
43
44      @FindBy(name = "activityIntervals[2].endTime")
45      WebElementFacade saida_field_2;
46
```

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

4.3 ARQUITETURA

A arquitetura do projeto foi definida de acordo com os padrões do *framework* Serenity BDD, que separa o código em um pacote principal chamado “src.test.java.meubridge.ua.cucumber”, onde ficam os arquivos java e um segundo pacote chamado “src.test.resources.features”, onde ficam os arquivos das funcionalidades e cenários. Além disso, há uma pasta denominada “*webdriver*” na qual ficam os executáveis que representam o *browser* que realiza a execução do teste, que no caso utilizado é o *ChromeDriver*.

Para a instalação do Serenity BDD foi utilizado o Maven, que é uma ferramenta de automação de compilação e instalação de projetos escritos em Java, utilizada para construir a estrutura do código que será executado. Para tanto foi criado um arquivo “pom.xml”, com as especificações de todos os frameworks que precisam ser instalados para a execução do projeto e de suas dependências, como por exemplo o JUnit, que é o motor de execução dos testes e o AssetJ, que é uma biblioteca open source para as asserções dos testes, ou seja, a verificação dos resultados do teste de acordo com o que é esperado.

Figura 14 - Estrutura hierárquica dos arquivos e códigos

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

Os *drivers* são configurados em um arquivo denominado “serenity.conf”, em qual também estão as configurações da URL de base do sistema que será testado, bem como as URLs das páginas que contemplam o módulo do escopo deste estudo de caso. Na figura acima é possível visualizar a estrutura dos arquivos e códigos.

4.4 EXECUÇÃO

Um caso de teste pode ser representado por um cenário, como também um cenário pode representar vários casos de teste usando uma estrutura chamada “esquema do cenário”, que utiliza um conjunto de exemplos para representar valores de entrada e saídas diferentes para um mesmo cenário. Na “Figura 15”, o cenário “Incluir Registro Diário com horas determinadas” possui dois conjuntos de dados diferentes, que serão executados independentemente, ou seja, será executado o cenário uma vez para o primeiro conjunto de dados e uma segunda vez para o segundo conjunto de dados.

O teste quando em execução utiliza o framework Selenium para realizar os passos dos cenários documentados, acessando o HTML da página através de um conjunto de identificadores como o ID do elemento, nome da *tag*, ou qualquer outro atributo único que identifique o elemento. Além disso, também é possível utilizar o *xPath*, que é uma linguagem de consulta que localiza o elemento por meio de uma expressão de navegação da página que indica o caminho até o mesmo.

Figura 15 - Código do arquivo registro_diario_editar.feature

```
#language:pt
@registrodiario @editar @all
Funcionalidade: Editar Registro Diario
  Como um usuario normal
  Eu gostaria de ir para o meu bridge
  A fim de editar registros de entrada

@ready @demo
Cenário: Editar Registro Diario
  Dado que eu realizo login na aplicação
  E eu seleciono a opção de editar
  Então eu devo poder alterar os campos de entrada e saída dos dias anteriores

@ready @demo
Esquema do Cenário: Editar Registro Diario com horas determinadas
  Dado que eu realizo login na aplicação
  E eu seleciono a opção de editar
  E eu insiro os valores para editar <entrada_0>, <saida_0>, <entrada_1>, <saida_1>, <entrada_2>, <saida_2>
  Então eu devo ver os campos de <total> e <saldo>
Cenários:
 | entrada_0 | saida_0  | entrada_1 | saida_1 | entrada_2 | saida_2   | total     | saldo     |
 | "0800"    | "1200"   | "1300"    | "1700"  | "1700"    | "1800"    | "09:00"   | "+09:00"  |
 | "0800"    | "1200"   | "1300"    | "1700"  | "1700"    | "1800"    | "03:00"   | "-01:00"  |

@pending
Cenário: Editar - Incluir Observação
  Dado que eu acesso a pagina principal
  E eu realizo o login
  E eu seleciono a opção de editar
  Quando eu clico em Observação
  E eu digito a observação "Horas Editadas"
  E eu clieo em Salvar
  Então eu devo ver a observação salva com "Horas Editadas"
```

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

5 ANÁLISE DOS RESULTADOS

Os resultados podem ser analisados por meio dos dados dos relatórios apresentados pela ferramenta Serenity BDD, que separa os testes de acordo com seus resultados e *tags* definidas para os mesmos antes da execução. Além disso, utiliza uma hierarquia de diretórios para organizar a documentação em formato HTML, que para ser disponibilizado para os envolvidos com os responsáveis pelo negócio do time, que envolve tanto os cenários automatizados, quanto os não automatizados em relação às funcionalidades documentadas.

Figura 16 - Serenity BDD: Requirements

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

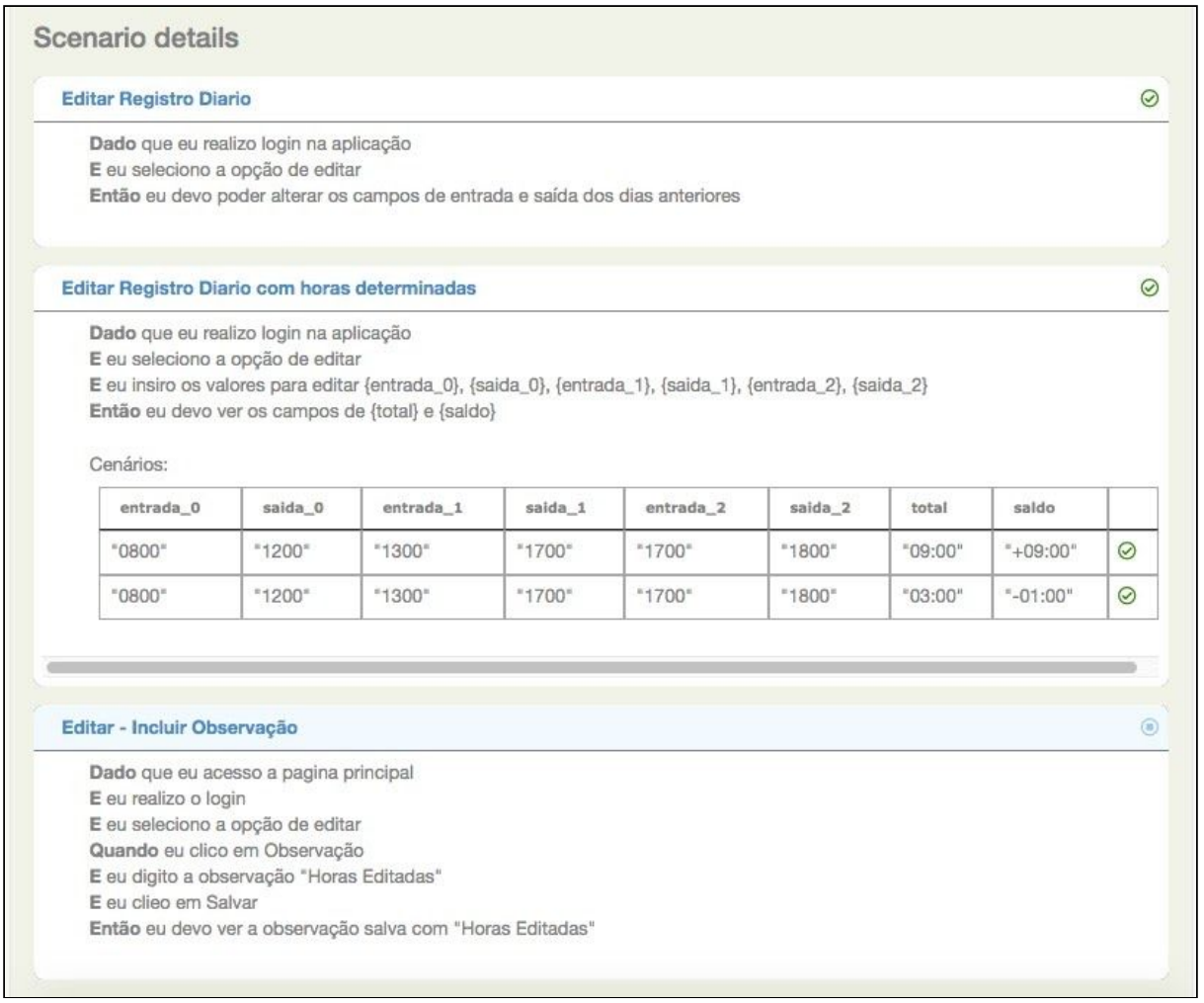
Ao selecionar um arquivo que referencia uma determinada funcionalidade é possível visualizar os resultados da execução da mesma. Como exemplo, a feature “Editar Registro Diário” teve seus resultados de acordo com o esperado, o que pode ser identificado pelo ícone de aprovação para os cenários:

Figura 17 - Feature: Editar Registro Diário

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

Ao clicar no botão de lupa é possível verificar os detalhes dos resultados da execução em relação aos cenários:

Figura 18 - Scenario details



Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

O Serenity BDD em caso de erro também específica o que ocorreu, como no exemplo abaixo o teste não encontrou o elemento de título dentro da página de registro diário:

Figura 19 - Log de teste com resultado diferente do esperado

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

5.1 VANTAGENS E DESVANTAGENS

5.1.1 Vantagens

Entre as vantagens de uma documentação viva escrita conforme o modelo proposto do presente estudo, está a facilidade na comunicação entre diversos papéis do time, independente do nível técnico, incluindo desde os clientes aos desenvolvedores, analistas e demais interessados no projeto. Além disso, reduz também problemas de interpretação, considerando que utiliza do conceito de criação de exemplos para o desenvolvimento da documentação.

5.1.2 Desvantagens

Uma das principais desvantagens é o teste intermitente, ou seja, no qual ocorrem interrupções que ocasiona a variação do tempo de execução de modo a impacto no resultado do teste. Esse problema não está diretamente ligado a documentação proposta, mas a qualquer teste automatizado, pois às vezes ao buscar por um elemento na página este não está mais presente.

O problema ocorre geralmente em testes automatizados de interface gráfica e tem como solução a inserção de um tempo de espera para o carregamento da página ou para os elementos de tela de forma individual, apesar dos *frameworks* geralmente já terem um tempo de espera implícito. Segue abaixo, um exemplo de método para espera de um elemento de tela ficar visível ou disponível:

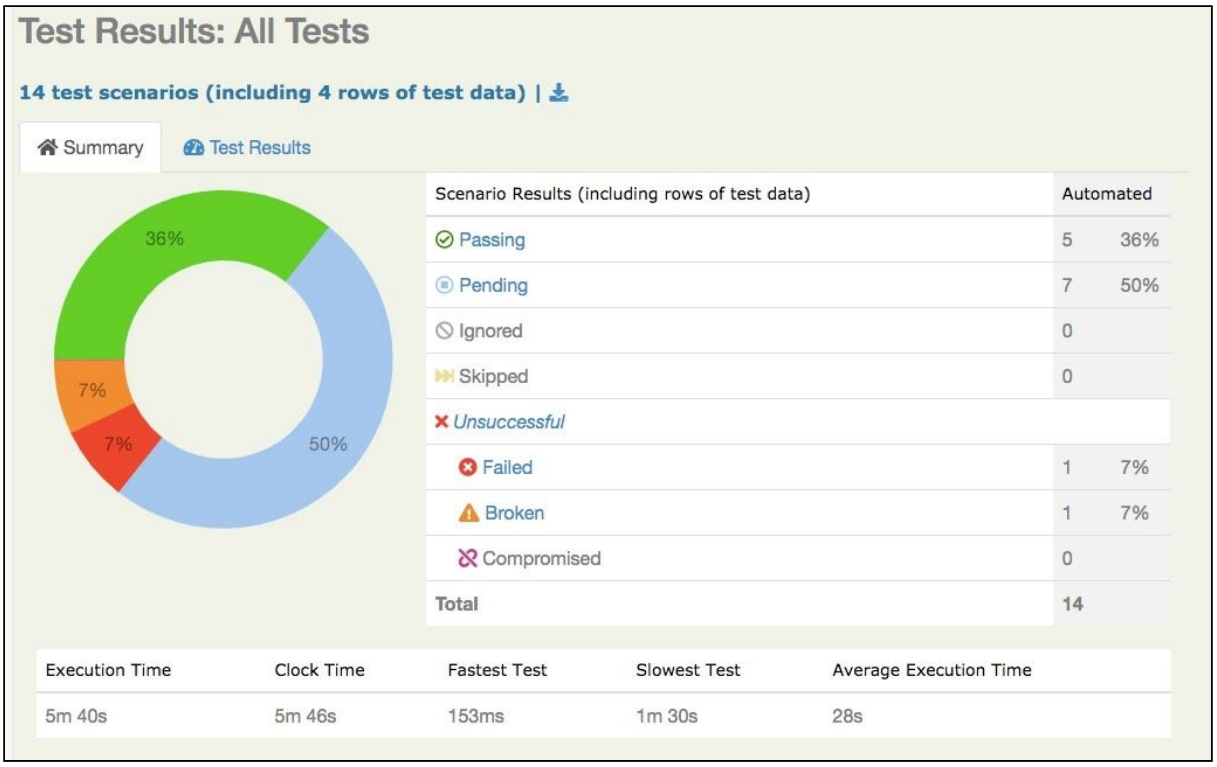
```
loginGmail.withTimeoutOf(20, TimeUnit.SECONDS).waitUntilVisible();
```

Uma observação importante sobre esse problema é que se o elemento de tela ficar visível em menos de 20 segundos, o teste passa para o método seguinte sem esperar os segundos restantes. Uma outra forma, pouco utilizada, é colocar o método *sleep*, que vai esperar um tempo mínimo de 20 segundos. Esta última forma não é indicada ser utilizada porque força esperar todos os 20 segundos até executar o próximo passo do teste.

5.2 COBERTURA DE REQUISITOS

A métrica de Cobertura de Requisitos pode ser medida através do Serenity BDD por meio do relatório da execução de todos os testes:

Figura 20 - Relatório do resultado de todos os testes



Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

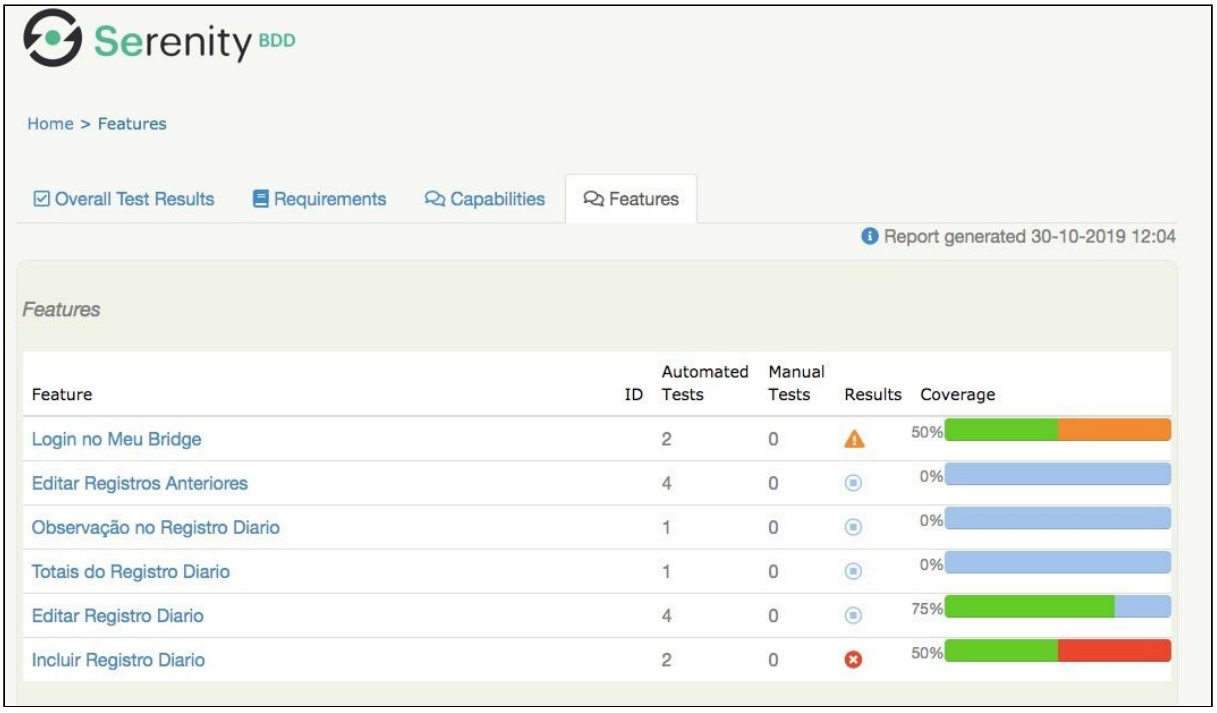
Neste relatório, podemos visualizar quais os testes automatizados que passaram e quais que falharam, bem como os que estão pendente de automação (*Pending*). Na coluna Automated por exemplo é possível visualizar a porcentagem das situações dos testes em relação ao todo.

A métrica de Cobertura de Requisitos no exemplo acima é de 50%, pois esse é o valor correspondente a proporção de cenários que foram documentados, mas que estão pendentes de automatização. A ferramenta calcula isso com base nas features descritas que possuem a *tag* @pending.

Outro dado interessante apresentado pelo Serenity BDD é a visualização da cobertura por funcionalidade, identificando se o teste que não foi automatizado é por

ser um teste manual. Essa situação ocorre quando o teste possui a *tag* @manual pela necessidade de ser realizado manualmente, decisão que não influenciando na métrica de *Pending*.

Figura 21 - Relatório de Cobertura por Funcionalidade



Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

6 CONCLUSÃO

O estudo de caso da utilização do Gherkin para automatização de testes e obtenção de métrica de Cobertura de Requisitos, possibilitou verificar que a linguagem é eficaz para gerar a rastreabilidade entre a documentação e os testes. Porém, necessita ser utilizada em conjunto com *frameworks* que gerem a execução desta documentação e relatórios sobre os resultados, como por exemplo o Cucumber e o Serenity BDD que foram aplicados no presente trabalho.

Os objetivos específicos estabelecidos para o estudo de caso foram atingidos, tendo sido desenvolvido alguns cenários em Gherkin para o módulo de “Registro Diário” do sistema Meu Bridge e se estendendo também ao módulo de Login, pois este era pré-requisito para acessá-lo.

A automatização dos cenários documentados foi realizada por meio da utilização do Cucumber e do Selenium, sendo que a segunda ferramenta já era utilizada pelo Laboratório Bridge para automatizar os testes. Algumas imagens dos códigos desenvolvidos e do processo de implementação, está disponível na seção de Implementação e Execução.

A obtenção da métrica da Cobertura dos Requisitos também foi atingida por meio da rastreabilidade da documentação criada pela utilização de tags específicas. O resultado pode ser verificado em Análise dos Resultados, no qual são apresentados os dados para a amostragem do escopo definido no trabalho.

Por fim, foram apresentados as vantagens e as desvantagens na adoção da prática, ficando a critério dos gestores do Laboratório Bridge e de outras pessoas interessadas na prática a avaliarem o custo benefício em relação aos esforços de

desenvolvimento e da alteração da cultura de documentação atual existente diante do estudo realizado.

6.1 CONSIDERAÇÕES FINAIS

Como considerações finais é importante salientar as dificuldades vivenciadas no desenvolvimento do presente trabalho para que isso possa ser avaliado durante a análise da adoção ou não do modelo de documentação proposto diante da necessidade da rastreabilidade para a obtenção da métrica de Cobertura de Requisitos.

Entre as principais dificuldades, podemos citar a documentação do Serenity BDD, pois apesar de ter bastante informação foi difícil encontrar um exemplo no site oficial em que o código estivesse funcionando adequadamente. Além disso, o exemplo que por ventura funcionava tinha versões do *framework* desatualizadas, o que resultava em falhas de combinação com as dependências.

Outro ponto que impactou bastante no tempo de desenvolvimento foi o fato de não existir uma forma padrão de estrutura definida para orientar a composição da arquitetura. As muitas possibilidades de configuração disponíveis para o *framework* acabam por dificultar.

6.1.1 Trabalhos futuros

Para fins de estudo de caso, apenas alguns requisitos foram desenvolvidos no modelo de documentação proposto, ficando como sugestão para trabalhos

futuros a revisão destes e a migração da atual documentação para estrutura de *User Stories* e cenários em Gherkin ou a adoção do modelo como complemento.

Além disso, os estudos promovidos durante o planejamento estratégico documentaram a realidade da equipe Automatizado em um contexto específico da área de qualidade do Laboratório Bridge, podendo ser utilizada como ferramenta para futuras mudanças no setor.

7 REFERÊNCIAS BIBLIOGRÁFICAS

[1] FALCONI, Vicente. **O verdadeiro poder**. Falconi Editora, 2003.

[2] RIES, Eric. **A startup enxuta**. Leya, 2012.

[3] CAROLI, Paulo. **Lean Inception**. Leanpub, 2018.

[4] Shook, John. **Gerenciando para o aprendizado**. Lean Institute Brasil, 2016.

[5] Spinola, Mauro de Mesquita; Berssaneti, Fernando Tobal; Lopes, Felipe Bussinger. **Gerenciamento da qualidade em projetos**. Elsevier, 2013.

[6] CEZERINO, Aparecida; NASCIMENTO, Fernando Paes. Utilização da técnica de desenvolvimento orientado por comportamento (bdd) no levantamento de requisitos. **Revista Interdisciplinar Científica Aplicada**, v. 10, n. 3, p. 40-51, 2016.

[7] CRISPIN, Lisa; GREGORY, Janet. **Agile testing: A practical guide for testers and agile teams**. Pearson Education, 2009.

[8] GRAHAM, Dorothy; VAN VEENENDAAL, Erik; EVANS, Isabel. **Foundations of software testing: ISTQB certification**. Cengage Learning EMEA, 2008.

[9] FRAGA, Arthur Henrique Della; MOLINARI, Nathan Junior. **Portal do Colaborador Meu Bridge: Módulo de Controle de Horas e Férias.** Trabalho de conclusão de curso (Bacharelado em Ciência da Computação) - Universidade Federal de Santa Catarina, Florianópolis, 2018.

[10] SAYÃO, Miriam; DO PRADO LEITE, Julio Cesar Sampaio. **Rastreabilidade de requisitos**. RITA, v. 13, n. 1, p. 57-86, 2006.

Estudo de caso da utilização de Gherkin para automatização de testes e obtenção de métrica de Cobertura de Requisitos

Larissa Taw Rumiana de Oliveira

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)

Caixa Postal 476 – 88.010-970 – Florianópolis – SC – Brasil

larissataw@gmail.com

Abstract. *This paper presents the study case of the use of Gherkin language as an alternative to the documentation traditionally used by the Bridge Laboratory. It introduces the difficulties and needs that drive market research to identify what could be improved within the automated testing quality process, as well as justifying the decision to opt for a pilot project for the use of Gherkin. In addition, it describes why the pursuit of metering requirements coverage and the actual results of introducing and automating the Gherkin language by creating automated test scenarios with Cucumber and Selenium within a limited system scope internally developed time tracking system.*

Key-words: Gherkin - cucumber - automated testing - traceability

Resumo. O presente trabalho apresenta o estudo de caso da utilização da linguagem Gherkin como alternativa a documentação tradicionalmente utilizada pelo Laboratório Bridge. Introduz as dificuldades e necessidades que direcionaram à uma pesquisa de mercado para identificação do que poderia ser melhorado dentro do processo de qualidade de testes automatizados, como também justifica a decisão por optar-se por um projeto piloto para o uso de Gherkin. Além disso, descreve o porquê da busca pela metrificação da cobertura de requisitos e os resultados reais da introdução da linguagem Gherkin e da automatização com ela obtida por meio da criação de cenários de testes realizados com o Cucumber e Selenium, dentro de um escopo delimitado do sistema de controle de horário desenvolvido internamente.

Palavras-chave: G herkin - cucumber - teste automatizado - rastreabilidade

1 INTRODUÇÃO

Estudo de caso sobre as dificuldades e necessidades da linguagem Gherkin para automatização e a obtenção da métrica de cobertura de requisitos, como também a avaliação dos resultados durante e após sua implementação. A principal finalidade foi contribuir com o Laboratório Bridge e as demais instituições na decisão por adotar ou não essa alternativa de documentação, baseando-se em dados reais sobre seus desafios e benefícios.

1.2 OBJETIVOS

1.2.1 Objetivos Gerais

O objetivo geral deste trabalho é apresentar os resultados da utilização de uma documentação criada com cenários em linguagem Gherkin, como também de apresentar os impactos e resultados na automatização dos casos de teste e na obtenção da métrica de cobertura de requisitos por meio da rastreabilidade.

1.2.2 Objetivos Específicos

- Desenvolver cenários em Gherkin para o módulo de “Registro diário” do sistema Meu Bridge;
- Automatizar os cenários com o Cucumber e o Selenium;
- Medir a cobertura de requisitos para o módulo automatizado;
- Analisar as vantagens e desvantagens na adoção da prática.

1.3.Organização do artigo

Na **seção 1**, é apresentado a introdução, descrevendo a motivação e justificativa, os objetivos (gerais e específicos) e como o artigo está organizado. Após isso, na **seção 2** descreve-se os principais conceitos envolvidos no estudo de cado, a seguir na **seção 3**, desenvolve-se o planejamento estratégico, no qual é apresentado o plano de ação. Na **seção 4**, está as informações referentes a implementação e execução. Na **seção 5**, é feito a comparação das vantagens e desvantagens com base na análise dos resultados. Por fim, na **seção 6** apresenta-se a conclusão do artigo, seguidas das referências utilizadas.

2 CONCEITOS

Para esclarecimento das diferenças entre os tipos de teste segue a definição de alguns conceitos, assim como uma breve introdução a metodologia de *Agile Testing* e os artefatos que serão utilizados para o desenvolvimento do estudo de caso (*User Storie*s e cenários em Gherkin):

2.1 NÍVEIS DE TESTE

A qualidade de um software pode ser avaliada em diferentes aspectos, sendo definida e comparada pelo ISTQB (2008) em diferentes níveis de teste de acordo com as especificações abaixo:

2.1.1 Teste de Componente/ Teste Unitário

O teste de componente ou teste unitário, podem ser conhecidos também como teste de módulo ou teste de programa e possuem como principal objetivo identificar defeitos nas funcionalidades dos softwares, verificando isoladamente cada componente, ou seja, os objetos, classes, etc.

2.1.2 Teste de Integração

Teste de integração busca verificar as interações entre diferentes componentes do sistema, avaliando como o sistema está funcionando em relação ao sistema operacional, ao sistema de arquivos e outros sistemas que possa ter alguma troca de informação.

2.1.3 Teste de Sistema

Teste de sistema verifica o comportamento do sistema, se está de acordo com o escopo definido e proposto antes do desenvolvimento, sendo realizado geralmente ao final do projeto para avaliar se atende as especificações definidas para o produto e tendo como objetivo encontrar o maior número de defeitos possíveis.

2.1.4 Teste de Aceitação

O teste de aceitação é na maioria das vezes de responsabilidade do usuário ou cliente final, apesar de envolver também outras partes interessadas. O objetivo principal é validar o sistema e estabelecer uma certa confiança.

2.1.5 Teste Funcional

O teste funcional é baseado nas funções do sistema, podendo ser realizado em qualquer nível de teste. São geralmente especificados em uma documentação e podem ser feitos com bases nos requisitos ou processos do negócio.

2.1.6. Teste de características do produto

Conhecido também como testes não-funcionais, pois buscam validar o quão rápido ou bem feito o sistema está executando determinada funcionalidade, sendo a validação do funcionamento tratado em outros níveis de testes descritos anteriormente. A seguir estão os tipos de testes que avaliam as características do produto:

2.1.6.1 Teste de Confiabilidade

Avalia o sistema em relação a tolerância a falhas, a capacidade de recuperação de um problema e a conformidade com o esperado.

2.1.6.2 Teste de Usabilidade

Avalia o quão fácil é um sistema em relação ao seu uso e a promoção da capacidade de aprender do usuário, como também a sua atratividade.

2.1.6.3 Teste de Eficiência

Avalia o comportamento do sistema, o quanto de recurso utiliza para desempenhar suas atividades.

2.1.6.4 Teste de Manutenibilidade

Avalia se o sistema é passível de ser analisado, alterado, testável, além de verificar a estabilidade e conformidade.

2.1.6.5 Teste de Portabilidade

Avalia o sistema em relação a adaptabilidade, instabilidade, coexistência, substituibilidade e conformidade.

2.1.7 Teste de Arquitetura

O teste estrutural é freqüentemente chamado de 'caixa branca' porque está interessado no que está acontecendo 'dentro da caixa', ou seja, no código. Pode ocorrer em qualquer nível de teste, embora tende a ser aplicado principalmente no teste de componente e de integração.

2.1.8 Teste de Regressão

O objetivo do teste de regressão é verificar se as modificações no software ou ambiente não causaram efeitos colaterais e que o sistema ainda atende aos seus requisitos, revisando testes que já foram executados anteriormente.

2.2 AGILE TESTING

Lisa Crispin e Janet Gregory (2009) define *Agile Testing* como um conjunto de atividades que giram em torno da escrita do código de produção dos testes e do desenvolvimento, de modo que todos da equipe são considerados desenvolvedores e devem estar focados em entregar um produto de qualidade.

2.2.1 *User Stories*

Conforme Cezerino e Nascimento (2016) os *User Stories* pertencem as metodologias ágeis de desenvolvimento, ajudando os *stakeholders* a partilhar os requisitos que devem ser implementados, deixando claro o porquê são necessários e quais valores de negócio apresentam.

Outros pontos importantes são as propriedades que cada *User Story* deve ter e que estão listadas a seguir:

1. Descritiva: deve ser possível extrair uma expectativa ou funcionalidade;
2. Estimável: deve ser possível estimar o esforço para a sua realização;
3. Testável: deve ser possível testar e verificar se está de acordo as expectativas dos *stakeholders*.
4. Categorizável: deve ser possível categorizar em grau de importância, priorizando de acordo com a necessidade.

2.2.1 *Gherkin*

Gherkin é uma linguagem executável, próxima a linguagem natural e que utiliza de palavras chaves para descrever processos de acordo com uma estrutura interpretável.

2.4 MÉTRICAS

Métricas servem para acompanhar o desempenho de uma equipe, com o objetivo de identificar possíveis problemas no processo de desenvolvimento das atividades ou para certificar-se de que estão sendo atingidos os resultados esperados.

> “Metrificar é o primeiro passo para o controle e eventualmente para a melhoria. Se você não consegue medir algo, você não consegue entendê-lo. Se você não consegue capturá-lo, você não consegue controlá-lo. Se você não consegue controlá-lo, você não consegue melhorá-lo. (A, Raphael Donaire; 2000, p. 8)”

2.4.1 Cobertura de Requisitos

De acordo com Sayão e do Prado Leite (2005) a Cobertura de Requisitos analisa a rastreabilidade entre requisitos e casos de testes, identificando quais não foram previstos e necessitam de testes complementares. Porém, no caso do presente estudo, está métrica avalia quais testes dos previstos na documentação não foram automatizados e estão pendentes de desenvolvimento.

3. PLANEJAMENTO ESTRATÉGICO

O planejamento estratégico envolveu a etapa de Acesso à Realidade, na qual ocorreram algumas análises e investigações dos processos de trabalho e dos resultados atuais, até a construção e execução do Plano de Ação após a definição do Estado Futuro desejável.

3.1.1 Acesso a Realidade

Para entender a situação foi necessário ir ao *Gemba,* que de acordo com John Shook (2016), significa “lugar real” e descreve onde acontece realmente o trabalho. Estar atuando dentro da equipe Automatizado criava a falsa sensação de

já estar no ambiente certo para acessar a realidade, porém muitas informações novas foram adquiridas após questionar como cada equipe atuava.

3.1.1.1 *Contexto*

A equipe Automatizado existe desde 2014, tendo gerado 8.463 testes automatizados para mais de 30 módulos do e-SUS AB. No início apenas a equipe Automatizado desenvolvia testes automatizados, tendo como base a documentação do sistema, realizando os testes do tipo caixa preta, ou seja, sem conhecimento do código. Porém, este cenário também passou por algumas alterações e tende a continuar mudando, pois agora os *testers* das equipes ágeis também criam testes automatizados, sendo a maioria teste de caixa-branca ou caixa-cinza.

3.1.2 Situação Atual

Inicialmente a equipe Automatizado era a única responsável pelo desenvolvimento dos testes automatizados, porém essa realidade mudou com a disseminação desse conhecimento para as equipes ágeis que passaram atuar com o papel principal nesta atividade. O Automatizado passou a estar encarregado por executar e realizar a manutenção dos testes, como também da aprovação dos *pull requests* da própria equipe e das demais, verificando se os testes desenvolvidos estão construídos corretamente.

Algumas equipes ágeis antes do desenvolvimento dos testes automatizados, realizam a criação de um documento com todos os possíveis casos de teste, enquanto que outras pensam apenas no cenário durante o desenvolvimento.

3.2 PLANO DE AÇÃO

O plano de ação foi delimitado para o módulo “Registro diário” do sistema “Meu bridge”. Abaixo segue as especificações do plano:

- Capacitação: tempo estimado em 40 horas-homem
    - Estudo sobre a utilização de *User Stories;*
    - Estudo sobre a utilização e desenvolvimento dos cenários em Gherkin;
    - Estudo da utilização do Cucumber e para automatização dos cenários.
- Implementação e execução : tempo estimado em 80 horas-homem
    - Criação de um novo modelo de documentação para o módulo definido para o estudo de caso, utilizando *user stories* e Gherkin;
    - Criação dos testes automatizados a partir dos cenários em Gherkin, utilizando o Cucumber como ferramenta;
    - Execução dos testes automatizados;
- Análise de Resultados: tempo estimado em 20 horas-homem
    - Avaliação das vantagens e desvantagens;Obtenção da métrica de Cobertura de Requisitos;
    - Considerações finais sobre a aplicabilidade de um novo modelo de documentação para o contexto do laboratório Bridge.

4 IMPLEMENTAÇÃO E EXECUÇÃO

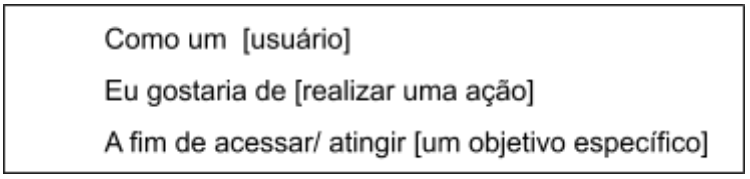
O processo da automatização dos casos de testes de uma documentação viva começam com a definição das *features*, ou seja, a descrição dos requisitos e funcionalidades para um determinado escopo, utilizando de uma linguagem passível de execução.

4.1 MODELO DE DOCUMENTAÇÃO

O modelo de documentação proposto é o composto por *User Stories* e cenários escritos em Gherkin. O *framework* escolhido e utilizado para a escrita dos cenários foi o Cucumber, que possui como linguagem original o inglês, porém também pode ser utilizado em outras línguas, como o português, desde que seja inserido no topo da *feature* a *tag* #language:pt.

A história do usuário deve ser escrita de acordo com a estrutura da figura abaixo e referenciada no arquivo de *feature* pela chave “Funcionalidade”.

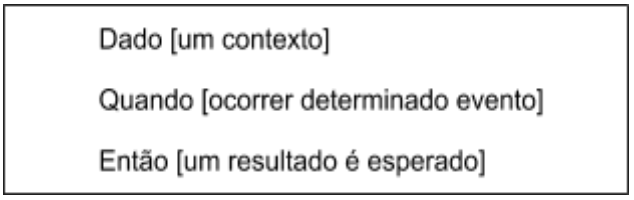
Figura 2 - Estrutura de User Story em Português



Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

Os cenários também possuem uma estrutura específica que deve ser referenciado no arquivo *feature* pela chave “Cenário”:

Figura 3 - Exemplo de Cenário em Português



Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

5 ANÁLISE DOS RESULTADOS

Os resultados podem ser analisados por meio dos dados dos relatórios apresentados pela ferramenta Serenity BDD, que separa os testes de acordo com seus resultados e *tags* definidas para os mesmos antes da execução. Ao selecionar um arquivo que referencia uma determinada funcionalidade é possível visualizar os resultados da execução da mesma.

Figura 3 - Log de teste com resultado diferente do esperado

Fonte: Elaborado por Larissa Taw R. de Oliveira, 2019.

5.1 VANTAGENS E DESVANTAGENS

5.1.1 Vantagens

Entre as vantagens de uma documentação viva escrita conforme o modelo proposto do presente estudo, está a facilidade na comunicação entre diversos papéis do time, independente do nível técnico, incluindo desde os clientes aos desenvolvedores, analistas e demais interessados no projeto. Além disso, reduz também problemas de interpretação, considerando que utiliza do conceito de criação de exemplos para o desenvolvimento da documentação.

5.1.2 Desvantagens

Uma das principais desvantagens é o teste intermitente, ou seja, no qual ocorrem interrupções que ocasiona a variação do tempo de execução de modo a impacto no resultado do teste. Esse problema não está diretamente ligado a documentação proposta, mas a qualquer teste automatizado, pois às vezes ao buscar por um elemento na página este não está mais presente.

6 CONCLUSÃO

O estudo de caso da utilização do Gherkin para automatização de testes e obtenção de métrica de Cobertura de Requisitos, possibilitou verificar que a linguagem é eficaz para gerar a rastreabilidade entre a documentação e os testes. Porém, necessita ser utilizada em conjunto com *frameworks* que gerem a execução desta documentação e relatórios sobre os resultados, como por exemplo o Cucumber e o Serenity BDD que foram aplicados no presente trabalho.

Os objetivos específicos estabelecidos para o estudo de caso foram atingidos, tendo sido desenvolvido alguns cenários em Gherkin para o módulo de

“Registro Diário” do sistema Meu Bridge e se estendendo também ao módulo de Login, pois este era pré-requisito para acessá-lo.

A automatização dos cenários documentados foi realizada por meio da utilização do Cucumber e do Selenium, sendo que a segunda ferramenta já era utilizada pelo Laboratório Bridge para automatizar os testes. Algumas imagens dos códigos desenvolvidos e do processo de implementação, está disponível na seção de Implementação e Execução.

A obtenção da métrica da Cobertura dos Requisitos também foi atingida por meio da rastreabilidade da documentação criada pela utilização de tags específicas. O resultado pode ser verificado em Análise dos Resultados, no qual são apresentados os dados para a amostragem do escopo definido no trabalho.

Por fim, foram apresentados as vantagens e as desvantagens na adoção da prática, ficando a critério dos gestores do Laboratório Bridge e de outras pessoas interessadas na prática a avaliarem o custo benefício em relação aos esforços de desenvolvimento e da alteração da cultura de documentação atual existente diante do estudo realizado.

REFERÊNCIAS

[1] FALCONI, Vicente. **O verdadeiro poder**. Falconi Editora, 2003.

[2] RIES, Eric. **A startup enxuta**. Leya, 2012.

[3] CAROLI, Paulo. **Lean Inception**. Leanpub, 2018.

[4] Shook, John. **Gerenciando para o aprendizado**. Lean Institute Brasil, 2016.

[5] Spinola, Mauro de Mesquita; Berssaneti, Fernando Tobal; Lopes, Felipe Bussinger. **Gerenciamento da qualidade em projetos**. Elsevier, 2013.

[6] CEZERINO, Aparecida; NASCIMENTO, Fernando Paes. Utilização da técnica de

desenvolvimento orientado por comportamento (bdd) no levantamento de requisitos. **Revista Interdisciplinar Científica Aplicada**, v. 10, n. 3, p. 40-51, 2016.

[7] CRISPIN, Lisa; GREGORY, Janet. **Agile testing: A practical guide for testers and agile teams**. Pearson Education, 2009.

[8] GRAHAM, Dorothy; VAN VEENENDAAL, Erik; EVANS, Isabel. **Foundations of software testing: ISTQB certification**. Cengage Learning EMEA, 2008.