**UM ESTUDO DE CASO PARA TESTES AUTOMATIZADOS COM *PYTHON* E *SELENIUM WEBDRIVER***

JOÃO PAULO D’ALMEIDA BONETTI

RAFAEL SILVA DALVAS

Orientação: Prof. Tatiane Ornelas Martins

**RESUMO**

A etapa de qualidade de software no desenvolvimento ou manutenção é imprescindível para o bom funcionamento e vida útil de um sistema. Assim sendo, a área que diz respeito aos testes automatizados desempenha um papel importante trazendo benefícios ao desenvolvedor e ao usuário. Com isso em vista, esse estudo de caso, por meio da ferramenta *Selenium* em conjunto com a linguagem de programação *Python*, visa fornecer uma etapa de qualidade a um sistema implementado sem a mesma, evidenciando os ônus de sua falta, bem como, obter resultados referentes às vantagens e desvantagens das tecnologias utilizadas. Dessa forma, foi demonstrado nesse estudo as etapas para obtenção dos resultados, notabilizando seus obstáculos relacionados às condições do sistema proposto. Portanto, concluiu-se que testes automatizados podem sempre contribuir para uma maior confiabilidade de uma aplicação além de trazerem ganhos à equipe de desenvolvimento.

**Palavras-chave:** Testes Automatizados. Qualidade de Software. *Selenium Webdriver. Python.*

***ABSTRACT***

*The stage of the software quality in developing and maintenance are essential for a well functioning system in it`s life cycle. Thus, the area that concerns automated testing, plays in an important role, and it brings benefits to the developer and user. With this in perspective, this case study, through the Selenium tool, together with the Python programming language, aims to provide a step of quality to a system implemented without it, evidencing the problems of its absence, as well as getting results regarding the advantages and disadvantages of the technologies that were used. So, it was demonstrated in this study all steps to get to results, evidencing its obstacles connected to the conditions of the proposed system. So, it was concluded that these automated tests can add a greater reliability to an application, besides bringing gains to the development team.*

***Keywords:*** *Test Automation. Software quality. Selenium Webdriver. Python.*

**1 INTRODUÇÃO**

Um teste de software se baseia em sua execução com o intuito de levantar informações, problemáticas e comportamento do mesmo. Com isso, a demarcação se esse atingiu os resultados previstos e funcionou corretamente conforme esperado, é favorecida.

O teste de software é uma parcela de todo o processo de qualidade de software. A última é definida por Bartié (2002, p.16) como: “[...] um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos de produtos, prevendo e eliminando defeitos”.

Dentro de testes de software, podemos abordar a automação dos mesmos. Essa se baseia em controlar e repassar para o computador a execução desses que seriam feitos manualmente. Seus benefícios em relação aos manuais incluem agilidade, que por consequência, geram um maior volume de testes e confiabilidade.

Ao nos referirmos às formas de teste automatizados, podemos listar diversas ferramentas que proporcionam ao desenvolvedor a possibilidade de realizá-los de forma rápida e simplificada, porém, a priori, iremos nos restringir aqui à ferramenta escolhida para a aplicação de nossos testes: o *Selenium*.

Esta ferramenta de automação permite ao usuário desenvolver *scripts* que, ao serem integrados, realizam execuções diretamente no navegador (no ambiente real da aplicação), utilizando-se do suporte à automatização nativa do mesmo. Todavia, a escolha do *Selenium* para nosso estudo de caso, trata-se não somente de sua versatilidade e facilidade de uso, mas também devido ao fato da não necessidade de acesso ao código fonte de aplicação, por se tratar de testes funcionais[[1]](#footnote-1).

Com base na premissa de que existem poucas informações disponíveis em relação ao tema, este tratado pode vir a ser alvo de pesquisas similares, aprofundando os conhecimentos de um contexto. Assim, em face do sistema estudado e ferramentas escolhidas para a abordagem dos testes de software, este estudo mostra o real proveito dessas ferramentas, beneficiando diretamente futuros estudos e abordagens semelhantes aos aqui apresentados.

No que diz respeito à linguagem de programação, foi utilizada a tecnologia *Python*, essa caracterizada por ser limpa e entendível, acarretando em um código mais compreensível dos testes automatizados.

O sistema iNtegra, utilizado pela comunidade acadêmica do Instituto de Ciências Exatas (ICE) da Universidade Federal de Juiz de Fora (UFJF), não possui testes, dificultando a manutenção do sistema. Assim, uma nova funcionalidade ou correção dispõe do risco de fazerem-se novos erros além da dificuldade de detecção dos mesmos.

Sabendo que além de possuirmos uma ferramenta e linguagem de programação condizentes, é de suma importância a definição de uma metodologia, em outras palavras, um padrão de projeto, para que nossos scripts de teste sejam desenvolvidos. Portanto, o emprego de *Page Objects*, tem como proposta a utilização do *Design Patterns* de nome *Page Objects Model* (POM), em suma, estrutura-se por um repositório de objetos com seus respectivos elementos para cada uma das páginas *Web*. Assim, deve haver uma classe[[2]](#footnote-2) correspondente para cada uma das mesmas.

Portanto, os objetivos envolvem aplicação dos testes automatizados para todas as funcionalidades do módulo de eventos, a partir da saída dos dados inseridos, disponibilizando um estudo de caso prático a respeito dos métodos de teste aplicados, bem como, seus resultados para a aplicação, avaliando o impacto de testes automatizados nela.

**2 JUSTIFICATIVA**

Segundo Delamaro, Andrade e Junior (2017), “negligenciar as atividades de teste, muitas vezes, pode remeter à produção de software de má qualidade e prejuízos econômicos”, acarretando, assim, em uma aplicação com baixa confiabilidade em seus processos. Considerando essa afirmação em conjunto com o duradouro tempo de atividade do sistema iNtegra (2009), há de se prever a exorbitância de possíveis defeitos e desproporções do mesmo.

Conforme exposto, o sistema iNtegra não passou por etapas de qualidade ou similares. Por suas altas possibilidades, a aplicação de testes manuais torna-se, em partes, irrealizável. Assim, um estudo de caso com base em testes automatizados realizado em um módulo específico, no caso o de eventos, pode proporcionar uma etapa de qualidade de software na aplicação. Como consequência disso, é viabilizado apontar os erros e desconformidades do módulo, além de, notoriamente, nos permitir observar como o *Selenium* pode auxiliar nos futuros desenvolvimentos do iNtegra.

Utilizando a frase de Kent Beck, criador do modelo de desenvolvimento de software *Extreme Programming* e *Test Driven Development* “Qualquer funcionalidade que não possui testes automatizados simplesmente não existe.” (apud CHEQUE, 2008, p. 43). Com isso, a apresentação de tais testes funcionais trazem a perspicácia da relação tênue entre os problemas detectados e a qualidade do software estudado.

**3 FUNDAMENTAÇÃO TEÓRICA**

Durante o processo de desenvolvimento de um software, ao se aplicar ações de Engenharia de Software, podemos definir o processo de aplicação de testes como a parte que requer maiores esforços. Especialmente se o mesmo for executado de forma informal o que, por consequência, pode acarretar em seus resultados (PRESSMAN, 1982).Quando negligenciado o planejamento, as chances de insucessos aumentam exponencialmente na detecção de possíveis problemas, o alto custo que será gerado pela inaplicabilidade e irredutibilidade do modelo serão altamente rigorosos (BARTIÉ, 2002).

A fim de garantir a devida aplicação dos testes, durante os anos de estudo e aprimoramento, foram criadas diversas técnicas distintas para sua realização. Porém, as diferentes técnicas de teste de software devem ser complementares, ou seja, servir como forma de traçar a estratégia que melhor corresponde à qualidade, eficácia e baixo custo (DELAMARO, 2013; MALDONADO, 2013; JINO, 2013).

Uma dessas técnicas seriam os testes automatizados que, apesar de não serem primordiais, uma vez que testes manuais também podem funcionar, dão maior manejo de tempo para o profissional da área e abrangem a qualificação e agilidade de entrega do mesmo aos desenvolvedores (WAZLAWICK, 2019), ocasionando maior qualidade, eficácia e baixo custo, pilares esses citados no parágrafo anterior.

Portanto, podemos dizer que a soma dos fatores manifestos acima garantem maior produtividade aos desenvolvedores, pois ao programador realizar a aplicação de testes automatizados, garante-se que seu tempo só será despendido durante a escrita dos mesmos. Desta forma, obtém-se gasto mais eficiente de tempo em comparação a outros métodos. (ANICHE, 2015).

**4 TRABALHOS RELACIONADOS**

Com intuito de encontrar uma forma mais assertiva de realizar os testes de forma automatizada, foi feito um estudo se baseando no fato de que as ferramentas de testes automatizados, metodologias e processos já usuais pelos mesmos eram mais utilizados para casos de uso específicos dentro da empresa (CARVALHO, 2010). Desta forma, visando a qualidade como o objetivo principal do estudo de inovar os processos de negócio internos da empresa, efetuando os testes individualmente, que até então eram previamente definidos e realizados manualmente, o que gerava uma demanda de mão de obra e processos muito alta, que refletia diretamente nos gastos. No estudo disposto, assim como no trabalho citado, tem-se como o princípio primordial a qualidade do Software, o uso de ferramentas para a realização de testes automatizados com *Selenium* que foi utilizado para o levantamento de informações sobre o sistema e na validação da qualidade do mesmo. Portanto, prezando pela qualidade, os testes são de suma importância para o software e assim o desafio de automatizar os casos de teste com o auxílio das ferramentas citadas, com o intuito de analisar as informações levantadas durante o processo e a possibilidade de uma *DashBoard* para auxiliar na exemplificação dos erros gerados pelo Software.

Um trabalho que pode ser citado, é o de Romanini e Sotto (2019), onde é analisada a relevância e benefícios de testes automatizados em projetos usando para estudo, também, o Selenium. Assim como no dos autores, esse estudo de caso aborda as definições de testes automatizados e uma explicação técnica das ferramentas utilizadas, como o *Selenium*. No artigo é contextualizado a automação de testes de software e aborda quando eles devem ser aplicados, além de expor fases técnicas das ferramentas usadas, dentre elas o *Selenium WebDriver*. Os resultados contemplam a maior qualidade de um software que passou por fase de testes e a maior eficácia e confiabilidade quando o último é automatizado. Contudo, evidencia que o momento de automatizar testes cabe ao responsável e não deve ser uma regra. O estudo, também, realça os incomensuráveis rendimentos que o *framework* proporciona e o quão completo é. A discrepância entre os dois trabalhos se dá uma vez que o de Romanini (2019) e Sotto (2019) utilizam de cenários de testes com acesso ao código fonte e em situações mais genéricas.

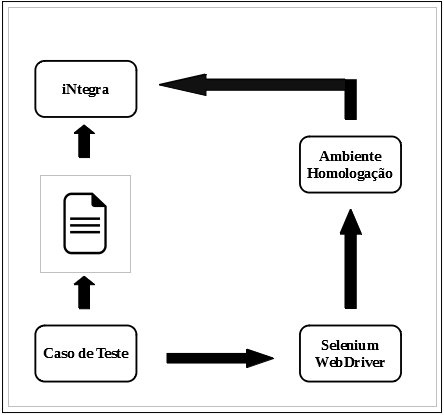
Em busca de uma análise mais abrangente do que somente a aplicabilidade técnica dos testes de software como abordado aqui, neste estudo é possível identificar os fatores primordiais na institucionalização das etapas de teste de software. Para que seja possível inseri-lo em um ambiente organizacional foi realizado um estudo de perspectiva sociotécnica (TOSETTO, 2008; BELLINI, 2008). Foram investigados, a partir de uma análise exploratória a uma empresa de sistemas de informação, obtendo entrevistas em profundidade com dez especialistas em qualidade e teste de software que, por meio das informações obtidas e cruzamento dos resultados da análise de conteúdo, foi feita uma relação entre os modelos organizacionais abordados, possibilitando a conclusão de qual seria o modelo ideal, em suas palavras “compreensivo e flexível”, de modo a ser válido e aplicável por qualquer organização em projetos de software.

**5 ARQUITETURA DO SISTEMA PROPOSTO**

O sistema proposto é composto pelos seguintes elementos, conforme pode ser observado na Figura 1.

1. iNtegra: sistema de domínio do módulo testado;
2. Ambiente de homologação: instância destinada a simulação de resultados na aplicabilidade dos testes, sem gerar impacto e/ou indisponibilidade do sistema em seu ambiente de produção;
3. *Selenium WebDriver*: ferramenta que dispõe de uma API para criar scripts de testes que são executados diretamente no navegador, utilizando o suporte à automação de cada um desses, ou seja, não necessita de acesso ao código fonte da aplicação;
4. Casos de teste: conforme definido em (IEEE 610), os casos de teste tratam-se do conjunto de valores de entrada, pré-condições de execução, resultados esperados e das pós-condições de execução, desenvolvidas para atender à condição dos testes aplicados;
5. Documentação: apresentação das evidências dos resultados provenientes dos casos de testes executados no ambiente de homologação.

Figura 1: arquitetura do sistema proposto.



**6 MÉTODO**

Por se tratar de um estudo de caso, a metodologia a ser utilizada será a de pesquisa exploratória que, em sua fase inicial, visa estreitar a relação com uma situação. Sendo aqui abordados os conceitos específicos para dispor uma abordagem prática das técnicas de testes de software apresentadas.

Assim, a pesquisa contempla as seguintes etapas:

1. pesquisa, levantamento, análise de técnicas de teste de software com *Python* a serem aplicados no sistema *iNtegra*;
2. pesquisa dos conceitos e abordagens que envolvem o uso de técnicas de teste funcionais de softwares, envolvendo *Python* como linguagem de programação baseando-se na ferramenta de teste *Selenium*;
3. estudo das funcionalidade e regras de negócio do sistema;
4. elaboração do documento de casos de testes a partir dos casos de uso;
5. desenvolvimento dos testes funcionais no sistema iNtegra;
6. aplicação dos testes de forma que seja possível efetuar uma coleta e análise dos resultados obtidos;
7. disponibilização dos resultados para a comunidade acadêmica, com o objetivo de disponibilizar um estudo de caso prático a respeito dos métodos de teste aplicados, bem como, seus resultados para a aplicação.

**6.1 Questões de pesquisa**

Levantar as problemáticas resultantes da não aplicação de testes funcionais anteriores à implementação da aplicação. Desta forma, os testes realizados deverão apontar as falhas presentes no módulo de Eventos utilizado pelo corpo docente do campus, bem como, apresentar a importância da utilização de ferramentas de teste no processo de desenvolvimento de software.

**6.2 Resultados esperados**

1. Realizar uma demonstração prática que sirva de referência para futuros estudos relacionados;
2. Demonstrar a importância da aplicação de testes automatizados no processo de desenvolvimento de um software*;*
3. Publicação dos resultados em revistas, congressos, simpósios e eventos específicos sobre o tema no país.

**7 CONSTRUÇÃO E EXECUÇÃO DOS TESTES**

A partir das premissas expostas no artigo, foi feito um estudo relacionado ao iNtegra, para entendimento de sua finalidade e de suas regras de negócio. O estudo consistiu na análise do conteúdo presente no manual de usuário, navegação manual no ambiente de homologação com acesso administrativo.

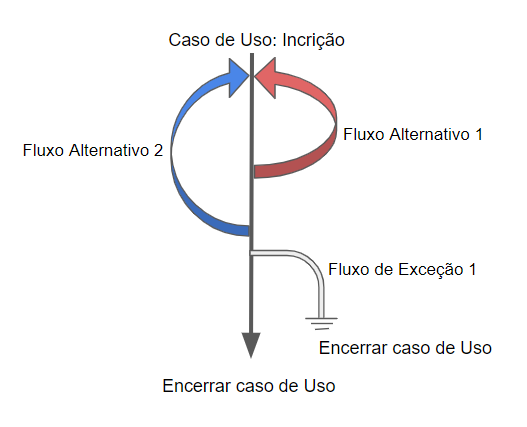
Também foram feitas entrevistas para coletar informações e aprofundar os conhecimentos obtidos por outros meios. Essas foram feitas com uma analista em tecnologia da informação da Universidade Federal de Juiz de Fora, tendo sido ela uma das responsáveis pela implementação do sistema. Desta forma, foi obtido o conhecimento necessário para a construção e execução dos testes funcionais.

**7.1 Documento de caso de teste**

Segundo Craig e Jaskiel (2002), caso de teste pode ser definido por: “uma condição particular a ser testada e é composto por valores de entrada, restrições para a sua execução e um resultado ou comportamento esperado”.

Para a elaboração do documento de caso de teste, foi utilizado um *guideline* disponibilizado pela UFPE, esse que trabalha derivando casos de teste de casos de uso. Ou seja, para cada fluxo de eventos de um caso de uso, que consiste em fluxo básico, fluxos alternativos e fluxos de exceção, existe um cenário de caso de uso que será usado como um caso de teste. Segue exemplo da estrutura citada de um caso de uso específico (caso de uso: Inscrição):

Figura 2: Fluxo de eventos de um caso de uso.



A partir dessas informações é derivado um caso de teste para cada fluxo de evento. Um caso de teste é definido por ações previamente definidas, bem como, o resultado a ser esperado que o sistema retorne. Como por exemplo o caso de teste “Ver Inscritos” derivado do caso de uso “Inscrição”:

tabela 1: Caso de teste de um cenário específico

|  |  |  |
| --- | --- | --- |
| **Cenários de Teste** | Ver Inscritos | |
| **Fluxos de Eventos** | Fluxo Básico | Fluxo de Alternativo 2 |
| **Ação** | Selecionar um evento com data de início superior a data do sistema, selecionar uma de suas atividades com vagas disponíveis e selecionar “Ver Inscritos”. | |
| **Resultado Esperado** | Sistema exibe a lista de interessados com os confirmados e em espera. | |

Com base nesses dados, torna-se possível realizar o desenvolvimento do caso de teste. Com esses parâmetros definidos conseguimos validar ou invalidar o mesmo.

O documento “Casos de teste - iNtegra” encontra-se disponível para acesso a partir do link[[3]](#footnote-3) disponibilizado.

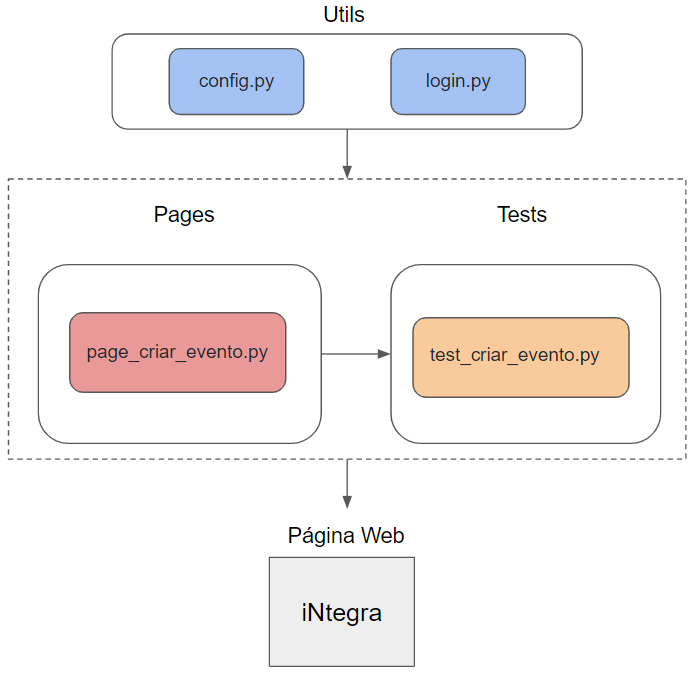
**7.2 Desenvolvimento dos testes**

O padrão de projeto *Page Objects Model* utilizado nesse projeto baseia-se na organização dos arquivos em diretórios de forma que há uma clara separação entre código de teste e código da página (localizadores e métodos da página em si). Na realidade desse estudo de caso, a utilização se dá por meio de três diretórios básicos, sendo eles: *Utils*, *Pages* e *Tests*.

* *Utils*: comporta as classes abstratas utilizadas em todo o projeto;
* *Pages*: comporta os códigos que dizem respeito aos localizadores e métodos de cada página;
* *Tests*: comporta os códigos de teste que chamam os métodos contidos nos arquivos localizados no diretório *Pages*.

Para exemplo, segue imagem da arquitetura proposta exibindo os arquivos da utils além dos arquivos de um caso de uso específico (Criar evento):

Figura 3: Exemplo da arquitetura *Page Objects Model*

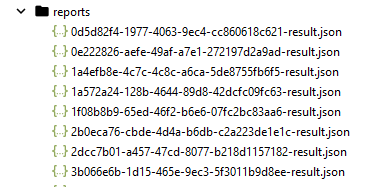


Todo o código que compõe o projeto se encontra no *Github* sendo, assim, possível fazer o *download* do mesmo através do link[[4]](#footnote-4) disponibilizado.

**7.3 Relatório de testes**

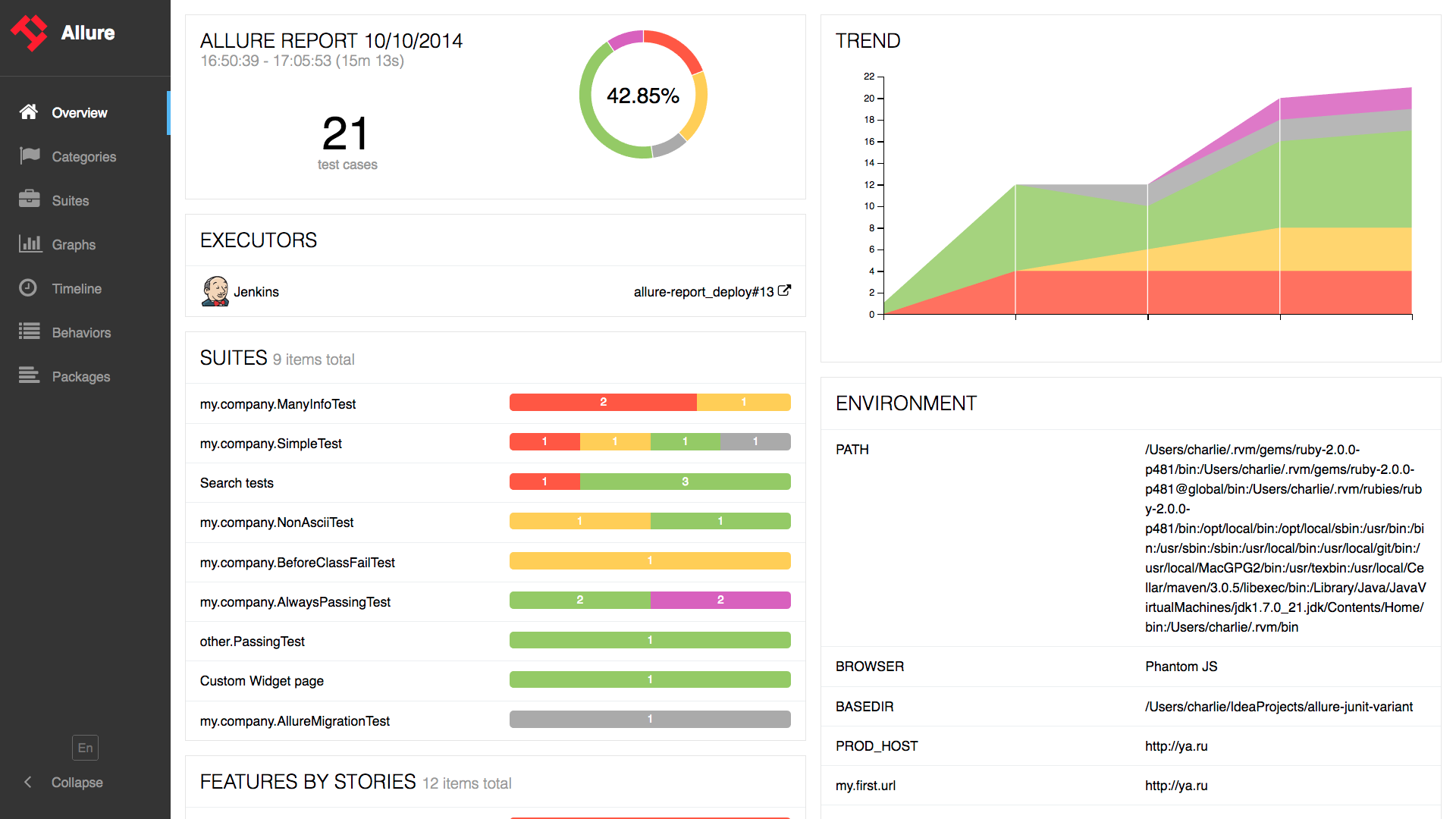
Para a coleta dos resultados de testes foi utilizado um *framework* denominado *Allure*. O *Allure* se define como uma ferramenta que, a partir dos resultados obtidos na execução do código em conjunto ao *Pytest framework*, é realizada a geração de pacotes *json* contendo as informações obtidas:

Figura 4: exemplo *json packages*



Esses pacotes, ao serem interpretados por um serviço local, geram resultados fornecendo ao usuário um quadro geral de relatórios que contém alguns tipos de gráficos e filtros para maiores detalhes.

Figura 5: Exemplo painel de controle *Allure*



Fonte: <https://docs.qameta.io/allure/>

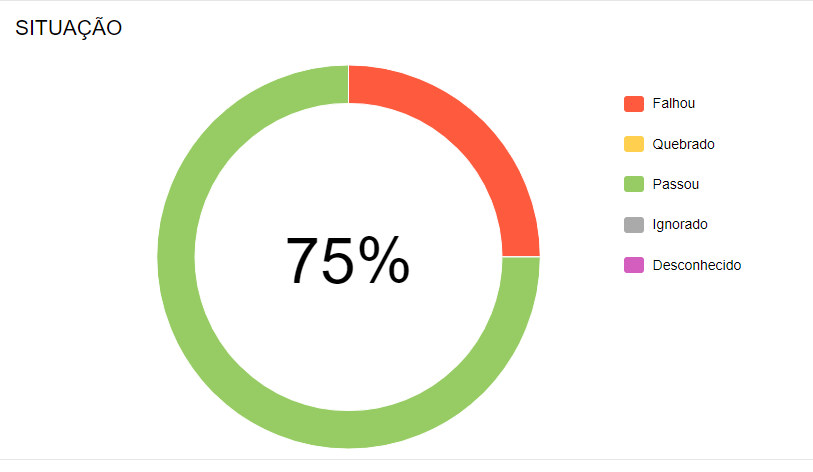
O relatório contendo o resultado de todos os casos de teste do sistema iNtegra, desenvolvidos e testados durante esse estudo de caso, encontra-se disponível para visualização a partir do link[[5]](#footnote-5)disponibilizado.

**8 RESULTADOS OBTIDOS**

Para a obtenção dos resultados referente ao estudo de caso, os *scripts* foram elaborados de forma a reconhecer a validade de seu resultado final. Desta forma, na eventualidade do resultado ser o definido como esperado, análogo ao descrito no documento de casos de teste previamente elaborado, o teste automatizado é classificado como “Passou” pelo *Allure*, caso contrário, como “Falhou”.

Após o desenvolvimento e implementação de todos os testes funcionais no sistema iNtegra, foi possível obter os seguintes dados estatísticos relacionados a validação ou falha de cada um dos casos de teste executados:

Figura 5: Resultado final dos testes no sistema iNtegra



Dentre os 64 casos de teste executados, em 48 (75%) deles o sistema apresentou o resultado esperado, aos restantes, o teste foi executado com sucesso, porém, o sistema não retornou o resultado esperado, portanto falharam.

**9 CONSIDERAÇÕES FINAIS**

O levantamento de dados para a elaboração dos documentos de testes demandou, desde o início, tempo e grande interação com responsáveis pelo iNtegra. Pois, o excesso de falhas no sistema, chegou a impossibilitar o acesso a algumas telas, que por consequência, a princípio, prejudicou a elaboração dos documentos de testes. Outra decorrência proveniente dessa falta de acesso a determinados recursos foi o estudo de certas regras de negócio, uma vez que ficava impossibilitado de completar o fluxo dessas. Portanto, foi necessário solicitar a correção de tais antes mesmo da elaboração dos casos de teste.

No que tange ao iNtegra, podemos citar a falta de adequação dele aos testes automatizados. Uma vez que não existe controle de qualidade na rotina de qualquer desenvolvimento do sistema, a harmonia com uma ferramenta com essa finalidade é precária. Malefícios quanto a isso incluem aumento no código de teste, pois, com isso, a captura dos elementos a serem interagidos pode se tornar defeituosa, caso haja alterações de desenvolvimento na tela, além de acarretar em um código mal apresentado. Tudo isso ocasiona um aumento de trabalho por parte de quem está desenvolvendo os códigos.

Um complemento válido aos testes funcionais aqui abordados, seria a realização de outras categorias dos mesmos em futuros estudos, a fim de levantar outros aspectos acerca do tema, como por exemplo, a aplicação de testes de performance, carga, estresse, estabilidade e regressão neste estudo de caso.

Com base nos resultados coletados e nos problemas enfrentados foi possível constatar que a falta de um controle de qualidade, mais especificamente testes no sistema, pode vir a gerar em demasiados erros que podem comprometer funcionalidades cruciais do sistema. É válido ressaltar que um sistema com demasiados erros acaba por vir a ser um de baixa confiabilidade, prejudicando, além de suas funcionalidades, a experiência do usuário.

A qualidade de software é definida como conformidade com requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas, que são esperadas em todos os softwares desenvolvidos profissionalmente. (PRESSMAN, 2006, p580).

Sendo assim, é importante salientar que o conceito de qualidade de software está diretamente ligado ao seu contexto, portanto para que esse estudo de testes automatizados possa ser replicado a outros trabalhos, aspectos aqui não abordados deverão ser considerados.

**REFERÊNCIAS**

BARBOSA, Tiago Ferreira. **Page Objects Pattern e por quê Aplicá-lo nos Seus Testes de UI (User Interface)**. Disponível em: <https://www.linkedin.com/pulse/page-objects-pattern-e-por-qu%C3%AA-aplic%C3%A1-lo-nos-seus-de-thiago/>. Acesso em 19 mar. 2020.

BARTIÉ, Alexandre. Garantia da qualidade de software. **Gulf Professional Publishing**. 2002.

BURNS, David. Selenium 1.0 Testing Tools Beginner's Guide. **Packt Publishing Ltd**. 2010.

CARVALHO, Márcio Felipe Alves. **Automatização de Testes de Software**. 109 f. Trabalho de Conclusão de Curso (Tese de Mestrado). Programa de Pós-Graduação do Instituto Politécnico de Coimbra – Instituto Superior de Engenharia. Coimbra: ISEC. 2010. Disponível em:<http://files.isec.pt/DOCUMENTOS/SERVICOS/BIBLIO/teses/Tese_Mest_Marcio-Carvalho.pdf>. Acesso em: 03 abr. 2020.

CRAIG, R. D.; JASKIEL, S. P. **Systematic software testing**. 1. ed. Boston: Artech house, 2002.

GONÇALVES, H.N. **Geração de Testes Automatizados utilizando o Selenium.** Trabalho de Conclusão de Universidade de Pernambuco. Escola Politécnica de Pernambuco. Graduação em Engenharia da Computação. 2011.

IEEE STANDARD COMPUTER DICTIONARY. **A Compilation of IEEE Standard Computer Glossaries**, in IEEE Std 610 , vol., no., pp.1-217, 18 Jan. 1991, doi: 10.1109/IEEESTD.1991.106963.

MALDONADO, J. C. *et al*. **Introdução ao teste de software**. São Carlos, 23. 2004.

MOLINARI, L. **Inovação e Automação de Testes de Software**. 1. ed. São Paulo-SP. 2010.

MORAES, Marina Adriana Faria de. **Automação de Testes de Regressão e Smoke Test em aplicações Web Com o Selenium**. Disponível em: http://repositorio.unis.edu.br/handle/prefix/994. Acesso em: 19 mar. 2020.

PRESSMAN, R.; MAXIM, B. **Engenharia de Software.** 8. ed. McGraw Hill. Brasil.2016.

PRESSMAN, R.; **Engenharia de Software.** 6. ed. São McGraw Hill.

RANSTROM, R.; NIKOLAI, C.; MADEY, G. Automated Web Software Testing With Selenium. **REU Project Paper**, Notre Dame. 2010. Disponível em: https://pdfs.semanticscholar.org/2de9/efcb0ff7967ccaaf9b2a7aee46d7f9c6f99b.pdf. Acesso em: 02 abr. 2020.

ROMANINI, I. Z.; SOTTO, E. C. S. Selenium Web Driver na Evolução dos Testes Manuais. **Revista Interface Tecnológica**, *16*(2), 112-123. 2019. Disponível em: https://doi.org/https://doi.org/10.31510/infa.v16i2.627. Acesso em: 02 abr. 2020.

TOSETTO, M.; BELLINI, C. G. P. Gestão sociotécnica do teste de software em projetos de sistemas de informação. **JISTEM-Journal of Information Systems and Technology Management**, 5(2), 325-346. 2008.

WAZLAWICK, R. Engenharia de software: conceitos e práticas. **Elsevier Editora Ltda**. 2019.

1. Técnica de teste de *software* baseada na validação da saída gerada pelo sistema após a entrada dos dados especificados. O comportamento interno do código fonte de sistema não é levado em consideração. [↑](#footnote-ref-1)
2. Conceito da programação orientada a objetos, as classes são responsáveis pela abstração de um conjunto de objetos. [↑](#footnote-ref-2)
3. <https://issuu.com/rafaeldalvas/docs/casos_de_teste_-_integra> [↑](#footnote-ref-3)
4. <https://github.com/rafaeldalvas/python-selenium-webdriver> [↑](#footnote-ref-4)
5. <https://jovial-gates-b45047.netlify.app/> [↑](#footnote-ref-5)