Estudo de caso: utilizando ferramentas de testes no desenvolvimento do *software* Ants

Valdrei Salomão da Silva [1], Daniela Marques [1], André Constantino da Silva [1], Gustavo Bartz Guedes [1]

[1]Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - Campus Hortolândia
Hortolândia – SP – Brasil

valdreisalomao@hotmail.com

marquesdaniela, andre.constantino, gubartz@ifsp.edu.br

Abstract. *Software testing is a key element in quality ensurance. Tests are divided and allocated according to their goal. Functional tests validate that the system works according the specifications. Performance tests evaluate the behaviour of a system when submitted to increasing load. Therefore, to ensure greater efficiency during the tests, it is important to use tools that automate somehow these activities. This article presents a case study using Ants, an Android application to aid philanthropic institutions, including OpenSource tools to perform the functional automated tests with Calabash and Jmeter for performance tests on Web Service.*

Key words: Automation tests, Open Source, Functional TestJmetering, Performance Testing, software Testing.

Resumo. *O teste de software é um elemento fundamental na garantia da qualidade. Os testes são divididos e atribuídos de acordo com seu objetivo. Os testes funcionais validam que as funcionalidades previstas estão de acordo com as especificações. Já os testes de performance garantem que as aplicações se comportem de forma esperada quando submetidas a situações específicas de cargas. Assim, para garantir maior eficiência durante a execução dos testes, é importante a utilização de ferramentas que automatizam de alguma forma essas atividades. Este artigo apresentará um estudo de caso utilizando o Ants, um aplicativo para Android para auxílio de instituições filantrópicas, utilizando ferramentas OpenSource para realizar os testes automatizados funcionais com o Calabash no aplicativo Android e o Jmeter para os testes de performance no Web Service.*

Palavras chaves: Automação de testes, Testes Funcionais, Testes de Performance, Teste de software.

1. Introdução

A ideia do aplicativo *Ants* surgiu após análise do crescente número de organizações não governamentais (ONG) e sem fins lucrativos no Brasil (Rede Brasileira do Terceiro Setor, 2017). Apesar desse número crescente, segundo uma amostra de soluções para filantropia analisadas por PINTO (2017), a maioria destas soluções tem um carácter informativo

e nenhuma integra o uso de geolocalização como forma de fomentar a filantropia. O *Ants* tem como objetivo auxiliar a interação entre instituições filantrópicas e possíveis voluntários, dando mais notoriedade a instituições e seus respectivos eventos próximo a localização atual do usuário por uso de geolocalização.

Também foi levado em consideração na construção do aplicativo a mobilidade e acessibilidade. O aplicativo foi desenvolvido para a plataforma *mobile* utilizando *Android*, devido a popularização dos *smartphones* no Brasil (CAPELAS, 2017), e por ser uma plataforma livre.

O projeto do *Ants* foi realizado por alunos do curso de Tecnologia em Análise e Desenvolvimento de Sistemas e foi divido em 3 sub-projetos:

- Desenvolvimento do *Web Service*: elaboração e integração de componentes de persistência e autenticação consistindo em uma arquitetura orientada a serviço.
- Desenvolvimento do Aplicativo: elaboração de aplicativo para dispositivos móveis *Android*.
- Elaboração e execução de testes: criação de casos de teste com o intuito de reduzir defeitos e falhas no desenvolvimento do aplicativo.

Este artigo apresenta um estudo de caso onde é abordado o processo de criação e execução dos testes para a aplicação *Ants*, focando a garantia de qualidade. Foram realizados os testes automatizados funcionais para o aplicativo e o testes de performance para o *Web Service*.

Atualmente, a qualidade é um aspecto que deve estar presente em qualquer produto ou serviço, e com o *software* não é diferente. No processo de desenvolvimento de *software*, a fase de testes é a mais importante para garantia de qualidade. Nesta fase as funcionalidades desenvolvidas são validadas, verificando se atendem os requisitos especificados e se o *software* foi construído de maneira correta. Ainda dentro do quesito qualidade e considerando o desenvolvimento para *Android*, Tiago Matos (2012) diz:

> "[...] esta fase é de suma importância para o sucesso de qualquer *software*, já que o conceito de teste de *software* é geral para as mais diversas modalidades. Em ambientes móveis, como *smartphones* e *tablets*, deve-se obter uma maior atenção durante a etapa de testes, pois trata-se de uma tecnologia muito recente sem muitos estudos e experiências comparadas a outros ambientes, como *desktop*".

Este artigo está organizado da seguinte forma: na seção 2 será apresentado o referencial teórico utilizado; na seção 3 será mostrado a metodologia utilizada no estudo de caso; na seção 4, o estudo de caso do aplicativo *Ants*, onde será demonstrado o uso prático dos conceitos utilizando ferramentas de testes automatizados para realização dos testes funcionais e de performance e, por último, na seção 5 a conclusão.

2. Referencial Teórico

As seções decorrentes deste artigo irão descrever de maneira sucinta as definições e conceitos que irão servir de apoio para melhor entendimento sobre o estudo de caso realizado com o aplicativo *Ants*.

2.1. SCRUM

O SCRUM é um *framework* de desenvolvimento ágil dividido em ciclos, denominados *Sprints*, que geralmente compreendem um intervalo de tempo de uma semana a no máximo 15 dias. Em cada *Sprint* são codificadas um subconjunto das funcionalidades do sistema, descritas em alto-nível em uma lista chamada *Backlog* do Produto. Essas funcionalidades são descritas em histórias que indicam como uma ação deve ser realizada. Durante a *Sprint*, reuniões são realizadas diariamente para atualizar a equipe sobre a implementação dos requisitos a fim de garantir a entrega (SUTHERLAND e SCHWABER, 2012).

2.2. Qualidade de *software*

Durante o processo de desenvolvimento de *software*, qualquer tomada de decisão pode comprometer sua qualidade final. Desta forma, a somatória de todas as ações tomadas no decorrer do ciclo de desenvolvimento irá afetar o produto final. Para garantir um produto que corresponda as suas especificações é necessário atribuir um esforço em qualidade em todas as etapas do processo.

A qualidade de *software* é difícil de se definir, e mesmo que uma definição seja proposta definitivamente, a mesma sempre será questionada e debatida interminavelmente. Para Roger Pressman (2011, p. 360) a qualidade de *software* pode ser definida como: “uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para que aqueles que produzem e para aqueles que utilizam.”

Uma outra definição mais completa e que abrange os conceitos já citados, é definida por BARTIÉ (2002, p. 16) "Qualidade de *software* é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos".

2.3. Teste de *software*

Garantir que um *software* está funcionando exatamente como o especificado nos requisitos e detectar erros durante o desenvolvimento de uma aplicação antes dela estar em ambiente produtivo é uma finalidade dos testes de *software*. Esse processo está ligado a dois termos conhecidos como Verificação e Validação, onde para Morlinari (2008, p. 96) "Verificação é o processo de confirmação de que algo (o *software*) vai ao encontro das especificações. Validação é o processo de confirmação de que o *software* vai ao encontro dos requerimentos do usuário."

Os tipos mais comuns de testes são caixa-branca, caixa-preta e caixa-cinza, dentro desse conjunto de métodos são estendidas outras variações de testes que especificam tarefas dentro da construção de um *software*.

Os testes de aplicativos móveis possuem algumas peculiaridades como o tipo de conectividade utilizada, tais como 3G e *Wi-fi*, que varia em termos de velocidade e segurança. Os recursos dos aplicativos móveis ainda são inferiores comparados a *laptop*, principalmente no que tange ao consumo de bateria em função do aplicativo. Existe também uma diversidade de aparelhos fabricados por diferentes empresas (MUCCINI et al, 2012).

2.3.1. Teste de caixa-preta

O teste de caixa-preta é baseado na entrada e saída de dados do ponto de vista do usuário final. Tem por objetivo verificar se os métodos da aplicação estão se comportando exatamente como a especificação. É geralmente criado e executado por analistas de testes ou pelos próprios clientes. Este tipo de testes não requer conhecimento da estrutura interna da aplicação.

2.3.2. Testes funcionais

Validar o comportamento de um *software* e garantir que o mesmo execute de acordo com as especificações, é um dos principais objetivos dos testes funcionais. De acordo com YOUNG e PEZZÈ (2008, p. 182) "O Teste funcional, também conhecido como teste de caixa-preta ou teste baseado na especificação, denota técnicas para derivar casos de teste a partir de especificações funcionais. ”

Sendo assim, é necessário que haja um conhecimento das regras de negócio do *software* a ser testado, para que sejam criados documentos que mapeiem os casos de testes mais próximos das especificações. “Esses documentos descrevem o comportamento que o *software* deve assumir nos diversos cenários existentes para cada requisito de negócio"BARTIÉ (2002, p. 113).

Essa categoria de testes simula ações semelhantes às que um usuário final atribuiria durante a utilizando de uma aplicação. “Os testes serão aplicados sem que exista uma preocupação direta com o algoritmo interno do *software*, empregando apenas a utilização de valores de entrada e saída como referencial para a avaliação da conformidade do requisito."BARTIÉ (2002, p. 113).

2.3.3. Testes de performance

Acessar uma aplicação e a mesma possuir um tempo de resposta de acordo com esperado em diversas situações e condições é um desejo comum desde o desenvolvimento da aplicação até seu uso em ambiente de produção. Tendo isso como objetivo, é necessário testar a performance da aplicação. O teste de performance "em geral verifica se o tempo de resposta é o desejado para o "momento"de utilização da aplicação e suas respectivas telas envolvidas. Verifica o sistema sob condições reais de acesso simultâneo ou de carga de dados"MOLINARI (2008, p.183).

Visando o desempenho, nos testes de performance a aplicação é exposta a situações previstas de limites máximos de acessos e concorrência para verificar se atende os requisitos não funcionais definidos. "O critério de sucesso aqui estabelecido é empregar o volume de transações e tempo de respostas obtidos nos testes e compará-los com os valores-limites especificados"BARTIÉ (2002, p. 117).

2.3.4. Testes de regressão

"Alterações na versão do *software* podem influenciar nos formatos das entradas e saídas e os casos de testes podem não ser executáveis sem as alterações correspondentes"YOUNG e PEZZÈ (2008, p. 454), sendo assim é necessário reexecutar estes testes com intuito de garantir que as demais funcionalidades ou partes do *software* já desenvolvidas estejam funcionando corretamente. Segundo Roger Pressman (2016, p. 478) "o teste de regressão é a reexecução do mesmo subconjunto de testes que já foram executados, para assegurar que as alterações não tenham propagado efeitos colaterais indesejados."

A execução total destes conjuntos de testes nem sempre é uma das melhores estratégias, sendo assim de acordo com YOUNG e PEZZÈ (2008, p. 455) "a priorização da execução dos casos de testes determina a frequência da execução dos casos de testes, executando-se todos eventualmente, mas reduzindo a frequência daqueles que se julga serem, por algum critério, menos capazes de revelar falhas."

2.4. Testes automatizados

Os testes automatizados ajudam a evitar um trabalho manual excessivo durante a etapa de testes e permitem a reexecução dos testes rapidamente sempre que necessário, contribuindo assim para a qualidade do *software*. Segundo Paulo Bernardo e Fabio Kon (2008)

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

Testes de regressão são facilmente executados com o uso dos testes automatizados. A cada alteração no sistema é possível fazer um teste de regressão, ou seja, garantir que o que funcionava no aplicativo não sofreu nenhum impacto pela mudança.

Com a utilização de ferramentas é possível visualizar de maneira rápida e simples o resultado de uma execução de testes. De acordo com Reginatto (2017):

> "Através dos testes automatizados é possível obter um *feedback* mais rápido, por exemplo, visualizar quais testes passaram e quais falharam, em qual passo do teste teve erro e qual o motivo. Muitas ferramentas oferecem gráficos e relatórios onde é fácil analisar os resultados da execução de uma suíte de teste."

2.5. Ferramentas

Afim de auxiliar no gerenciamento e execução das atividades de testes do aplicativo *Ants*, ferramentas de testes automatizados e de controle de defeitos foram utilizadas.

2.5.1. *Calabash Android*

O *Calabash* (Calabash, 2017) é uma ferramenta *opensource* [1] utilizada para automação de testes voltados para aplicativos móveis para *Android* e IOS. A ferramenta *Cucumber* (Cucumber, 2017) é utilizado pela *Calabash* e possibilita a criação da escrita dos casos de testes em linguagem natural. Para desenvolver a automação dos passos dos casos de testes, o *Calabash* utiliza a linguagem de programação *Ruby* (Ruby, 2017).

2.5.2. Jmeter

O *JMeter* (Apache, 2017) é um *software opensource* que possibilita a criação e execução de testes automatizados de performance. Para a realização dos testes, o *JMeter* disponibiliza diversos recursos tais como controladores de requisições, controle de usuários virtuais (*threads*) e a partir dos resultados das requisições são gerados gráficos e tabelas para análise.

2.5.3. Mantis

O Mantis é uma ferramenta *opensource* para gerenciamento de defeitos onde é possível cadastrá-los, descrevendo o problema e os passos para sua reprodução, podendo ou não anexar evidências ao problema detectado. Todos os defeitos são registrados e atribuídos a alguém da equipe e permanecem nesse *status* até que o defeito seja Resolvido (alterado código e feito reteste) ou Rejeitado (defeito inválido).

3. Metodologia

O aplicativo foi desenvolvido de forma incremental utilizando a metodologia SCRUM. Os requisitos foram identificados após reunião feita com duas entidades filantrópicas que descreveram seus desejos, problemas e dificuldades. Baseado nessas necessidades, foi feita uma análise e documentados os requisitos em formato de histórias de usuário. Utilizou-se a ferramenta Trello (Atlassian, 2017) para compartilhamento de conteúdo entre os envolvidos no projeto.

Com foco na qualidade do aplicativo, os cenários de testes foram elaborados baseados nas histórias e serviram como guia para a criação e execução dos testes unitários durante o desenvolvimento. Os testes unitários foram criados e executados pelos próprios desenvolvedores.

Sabendo que os testes unitários são somente uma das etapas para garantia de qualidade, foram realizados outros testes: funcionais e performance. Os testes funcionais garantem que o aplicativo atende a especificação dos requisitos mapeadas nas histórias do usuário e que os testes de performance atende os requisitos não funcionais definidos.

Para os testes funcionais no aplicativo *Android* os mesmos foram elaborados e executados de maneira incremental conforme as funcionalidades do aplicativo eram de-

[1] *Opensource* é um termo em inglês que significa código aberto. Isso diz respeito ao código-fonte de um *software* que pode ser adaptado para diferentes fins. (Canaltech, 2017)

senvolvidas. Visando acrescentar valor nos testes e evitar esforços futuros, foi utilizada a ferramenta de testes automatizados *Calabash Android*.

Os testes de performance no *Web Service* tiveram como objetivo medir o tempo de resposta do servidor simulando cenários de acordo com as histórias desenvolvidas.

Defeitos podem ser encontrados durante a execução dos testes, registrar e mapear esses defeitos é uma etapa de suma importância para a qualidade da aplicação, é através deste controle que é possível saber o estado do mesmo e quem é responsável pelo defeito. Para o controle e gerenciamento de defeitos foi utilizada a ferramenta Mantis Bug Tracker (Mantis, 2017), assim sendo, todo defeito encontrado durante a execução dos testes no aplicativo e/ou *Web Service* era armazenado na ferramenta.

4. Estudo de caso

Para o estudo de caso será abordado todo o processo de desenvolvimento dos testes, desde a concepção do cenários de testes durante o processo de análise de requisitos, até a criação e execução dos casos de testes funcionais e de performance.

4.1. Requisitos e histórias de usuário

Baseado no *backlog* do produto, todos os requisitos especificados foram documentados em formato de histórias de usuário. A partir dessas histórias, os possíveis cenários de testes foram criados. Os mesmos foram elaborados levando em consideração cenários positivos e negativos, para garantir a cobertura de testes.

Para este estudo de caso será apresentada a criação e execução dos testes de uma das histórias. A medida que as *Sprints* avançavam, o processo se repetia. As histórias de usuário escolhidas para o estudo foram priorizadas por grau de relevância de acordo com a proposta do aplicativo *Ants* baseando-se nas principais funcionalidades que seriam utilizadas pelos usuários. Ao todo foram especificadas 11 histórias. A Figura 1 apresenta algumas dessas histórias. Para o estudo de caso dos testes funcionais foi usada a história "US2 - V1"referente ao cadastro de instituição. As histórias "US2 - V1", "US4 - V1"e "US7 - V1", respectivamente referente aos cadastros de instituição, oportunidades e usuários, serão utilizadas para os testes de performance.

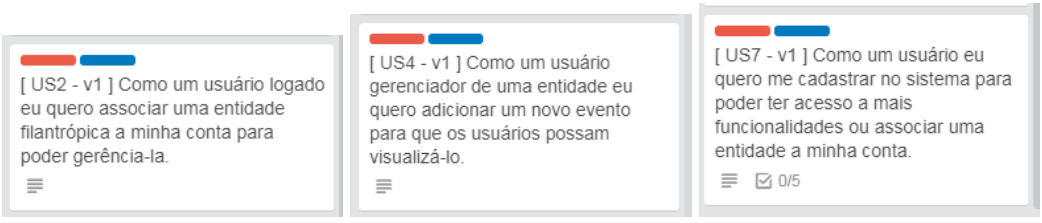

Figura 1. Histórias de usuário utilizadas no estudo de caso

4.2. Aplicação dos Testes funcionais - *Calabash Android*

Para cada *Sprint*, os testes funcionais eram criados e executados conforme as funcionalidades priorizadas. A primeira história priorizada foi a "US2 - V1"onde o usuário já cadastrado e conectado na aplicação seleciona a opção de associar instituição e preenche os dados obrigatórios para registrar essa entidade.

4.2.1. Criando a Estrutura dos testes

Após instalar e configurar a ferramenta, foi necessário criar a estrutura dos testes. Estas estruturas são um conjunto de pastas que possuem arquivos chaves para criação e escrita dos casos de testes e dos *scripts* [2] de testes. A estrutura é composta pela pasta raiz onde ficam armazenadas as pastas Funcionalidades (*features*) e Servidor de Testes (*test servers*):

- *Features*: estão os arquivos de extensão *.feature*, são os arquivos onde são escritos os cenários de testes em linguagem natural. Dentro da pasta *.feature* estão as pastas *Step definitions* e *Suport*.
  - *Step definitions*: contém os arquivos de extensão (.rb) e são neles que são escritos os *scripts* de testes utilizando a linguagem Ruby.
  - *Suport*: possui os arquivos de configurações que importa as bibliotecas necessárias para a execução dos testes.
- *Test servers*: possui o arquivo que salva as configurações de comunicação entre o aplicativo móvel e a ferramenta *Calabash*.

4.2.2. Desenvolvimento das features

Para o desenvolvimento das *features* é necessário levar em consideração os critérios de aceitação da funcionalidade. A seguir são apresentados os cenários escritos para a história "US2-V1":

1. Validar os campos da tela Associar Entidade;
2. Associar uma entidade com sucesso;
3. Validar se os dados foram registrados de uma entidade recém-registrada;
4. Pressione o botão Voltar enquanto preenche os campos e continue preenchendo até confirmar o registro;
5. Altere um campo de uma entidade já registrada e valide se o dados foram salvos;
6. Pressione o botão voltar enquanto preenche os campos e descarte as mudanças;
7. Não preencha um campo e pressione o botão confirmar;
8. Pressionar o botão confirmar sem preencher nenhum campo;

Um exemplo de implementação utilizando o cenário 2 com linguagem natural é estruturado conforme a Figura 2.

[2] Scripts são "roteiros" seguidos por sistemas computacionais e trazem informações que são processadas e transformadas em ações efetuadas por um programa principal. (PEREIRA, 2012)

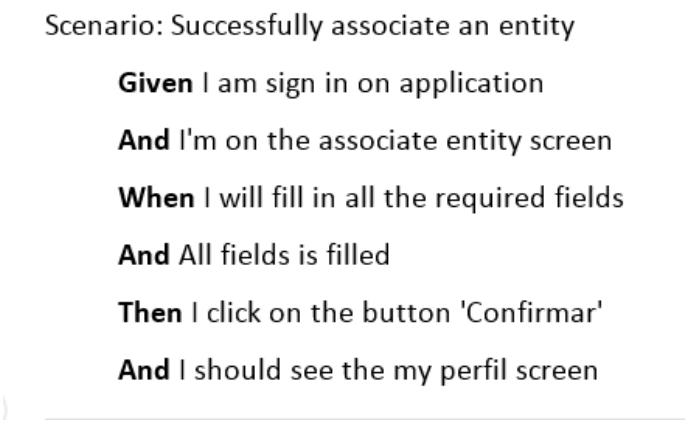

Figura 2. Estrutura de linguagem natural utilizada na *feature*

4.2.3. Desenvolvimento dos Steps Definitions

Ao executar o arquivo *feature* o mesmo gera uma estrutura com apenas os passos dos cenários de testes. Sendo assim, é necessário desenvolver os *scripts* de teste para que ao executar a *feature*, a mesma realize todos os passos diretamente no aplicativo, esses *scripts* são desenvolvidos no arquivo dentro da pasta *Step definitions*. Na Figura 3 é possível visualizar um exemplo de um dos passos do cenário 2 apresentado na seção anterior, onde no item 1 representa a estrutura criada logo após executar a *feature* e no item 2 é local de desenvolvimento dos *scripts* onde o qual é utilizado para criação e execução dos comandos que irão fazer toda interação com o aplicativo, os comandos são escritos na linguagem de programação Ruby.

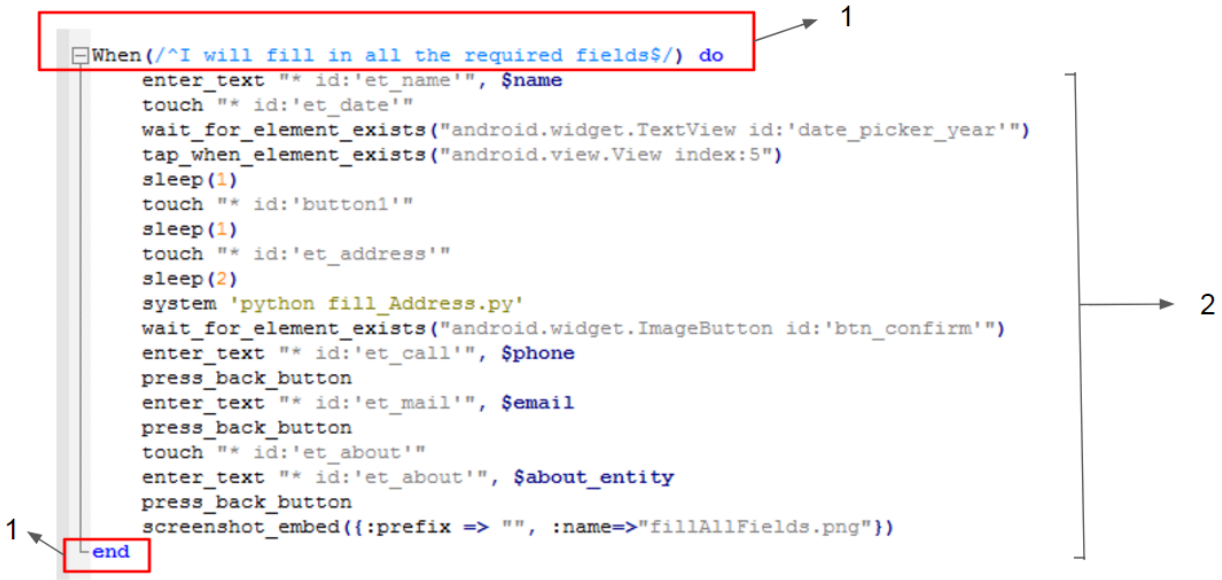

Figura 3. Estrutura do desenvolvimento do *script* de um dos passos do cenário 2

4.2.4. Execução e resultados

Após desenvolver o *script* é necessário executar novamente a *feature* afim de gerar um arquivo de reporte dos resultados do teste. O reporte é um arquivo html que contém todas as informações relevantes do teste. A Figura 4 mostra um exemplo do reporte do resultado da execução do cenário 2, os itens enumerados serão explicados logo abaixo:

1. Local onde é apresentado na primeira e segunda linha a quantidade de cenários e passos executados e passados, na terceira mostra o tempo total de execução do teste;
2. Local que indica através de cores o *status* da execução do teste, onde o mesmo fica verde quando todos os passos passaram e vermelho quando pelo menos 1 passo falha;
3. Nome da funcionalidade testada;
4. Nome do cenário testado;
5. Local onde é apresentado os passos do cenários, junto é mostrado as capturas de telas do aplicativo que evidenciam o passo testado. Quando um passo falha, é neste ponto que é apresentado o passo e a linha do arquivo de *scripts* que falhou;

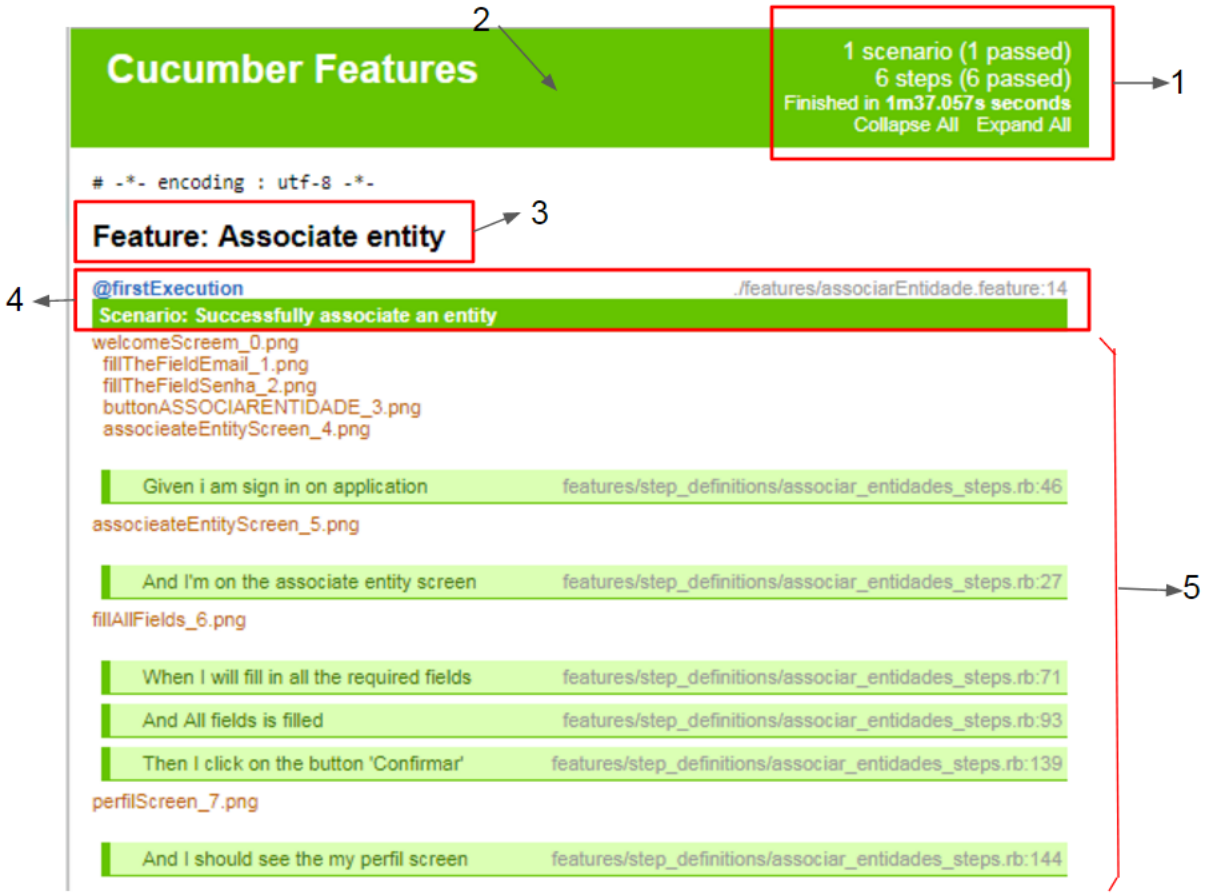

Figura 4. Reporte do resultado de execução do cenário 2

Finalizado toda a parte de configuração e criação dos cenários e *scripts* de teste, basta executar a *feature* para se obter um reporte final dos resultados da execução, para

execução final das duas histórias usuários apresentadas previamente foi obtido o seguinte resultado conforme a Figura 5 e Figura 6. Quando a aplicação sofre alguma alteração, a *feature* da funcionalidade é reexecutada para verificar se houve algum impacto em função dessa alteração. O mesmo acontece a cada *Sprint* executada, onde é feito um reteste das funcionalidades para garantir que a história adicionada não causou impacto no que já existia no aplicativo.

Figura 5. Resultado final da execução da *feature* "US2 - V1"

Figura 6. Resultado final da execução da *feature* "US4 - V1"

O processo de criação e execução dos testes se repetiram a cada *Sprint* rodada. Novas histórias eram adicionadas aos testes funcionais e as histórias de *Sprints* anteriores passavam por um teste de regressão para garantir que o funcionamento permanecia inalterado. A ferramenta *Mantis* foi utilizada para criação de defeitos nos casos de falha nos testes. Desta forma, ao cadastrar o defeito, o desenvolvedor era notificado e após arrumado o defeito, o mesmo era enviado para reteste até que fosse Resolvido. Na Figura 7 é possível visualizar a página inicial da ferramenta *Mantis*, onde é apresentado um resumo das atividades relacionadas ao controle e gerenciamento de defeitos já mapeados e resolvidos.

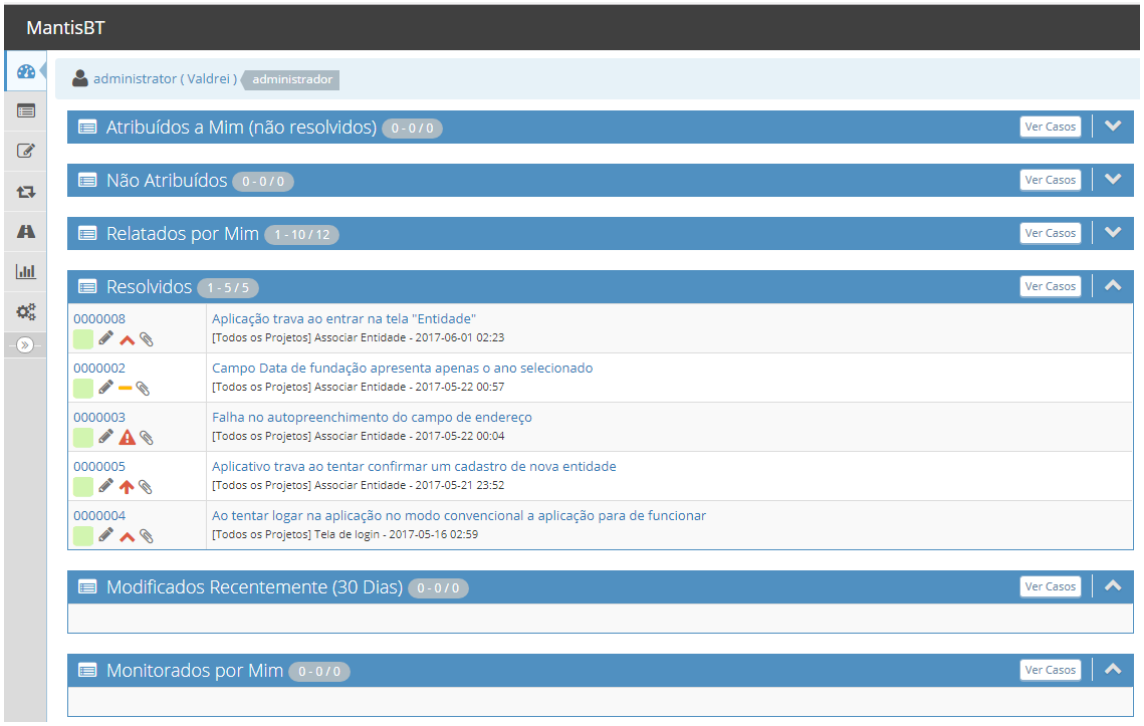

Figura 7. Visão da tela inicial da ferramenta Mantis

4.3. Aplicação dos Testes performance - *Jmeter*

Os testes de performance visam verificar o desempenho baseado nos requisitos não funcionais do sistema. Para este estudo de caso foi utilizada a ferramenta *Jmeter*. Os requisitos não funcionais foram obtidos a partir das expectativas dos membros do projeto a fim de especular uma carga inicial para aplicação. Determinou-se nessa primeira versão que o aplicativo suportasse ao menos 100 usuários concorrentes. Os testes de performance foram simulados em um servidor local, utilizando as configurações conforme a Figura 8. A latência de rede não foi considerada nos testes.

Configurações	
Modelo	Notebook Lenovo s400
Sistema Operacional	Ubuntu 17
Processador	Intel® Core™ i5-5200U
Memória	Memória Kingston 8GB 1333Mhz DDR3 CL9
Hard Disk	HD Seagate SATA 2,5´ 500GB 5400RPM SATA 6.0Gb/s
SSD	SSD Kingston 2.5´ 120GB A400 SATA III

Figura 8. Configurações da máquina utilizada para rodar o servidor

De acordo com Nielsen(2014), para que o usuário se sinta livre durante a navegação a resposta as suas ações devem ser respondidas em um período de no máximo um segundo e o tempo máximo para se manter a atenção do usuário no diálogo é no máximo 10 segundos.

A Tabela 1 expõe o número de usuários concorrentes por minuto para dois cenários utilizando as três histórias citadas anteriormente, cadastro de instituição, oportunidades e usuários. O Cenário 1 apresenta o número de usuários esperados por minuto simulando consultas no servidor. O Cenário 2 mostra o número de usuários concorrentes esperados para inclusão de instituições, usuários e oportunidades. Os testes foram executados simultaneamente com os dois cenários. Contudo, dado uma diferença relevante entre o tempo de consultas e inclusões, suas amostragens são expostas em figuras separadas.

Tabela 1. Quantidade de chamadas esperadas

Cenário 1	QTD/Minuto	Cenário 2	QTD/Minuto
Consulta de instituição	3	Cadastro de instituição	1
Consulta de usuário	3	Cadastro de usuário	3
Consulta de oportunidade	6	Cadastro de oportunidade	2

Durante o período de execução do teste, um minuto, a quantidade de usuários concorrentes foi incrementado gradualmente obtendo a carga máxima para o mesmo de acordo com os cenários descritos acima. As Figura 9 e Figura 10 mostram respectivamente a carga de usuários concorrentes no período para consultas e inserções respectivamente.

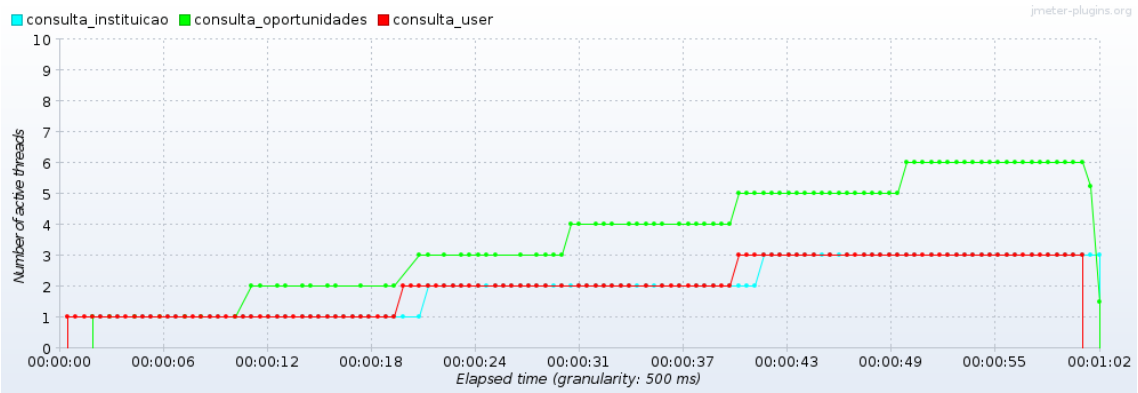

Figura 9. Número de threads executadas simultaneamente - Consultas

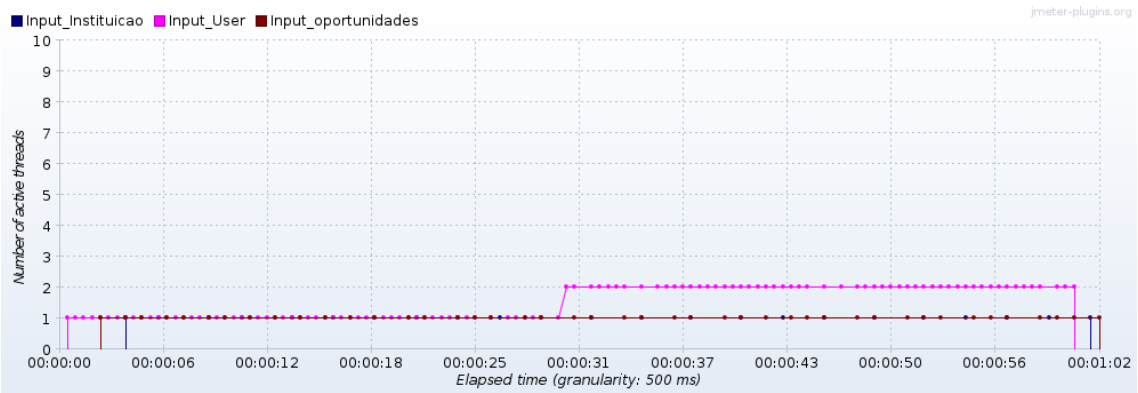

Figura 10. Número de threads executadas simultaneamente - Inserções

A Figura 11 apresenta relação entre o tempo de execução e o tempo de resposta para as consultas feitas pelos usuários concorrentes exibidos na Figura 9. Observou-se que os objetos que detêm relação de chave estrangeira no banco de dados como instituição e oportunidade tem um tempo de resposta superior ao objeto usuário que não detêm nenhuma relação de chave estrangeira para com suas informações compartilháveis. Observa-se que o tempo de resposta mais alto permanece dentro do requisito não funcional definido para o aplicativo.

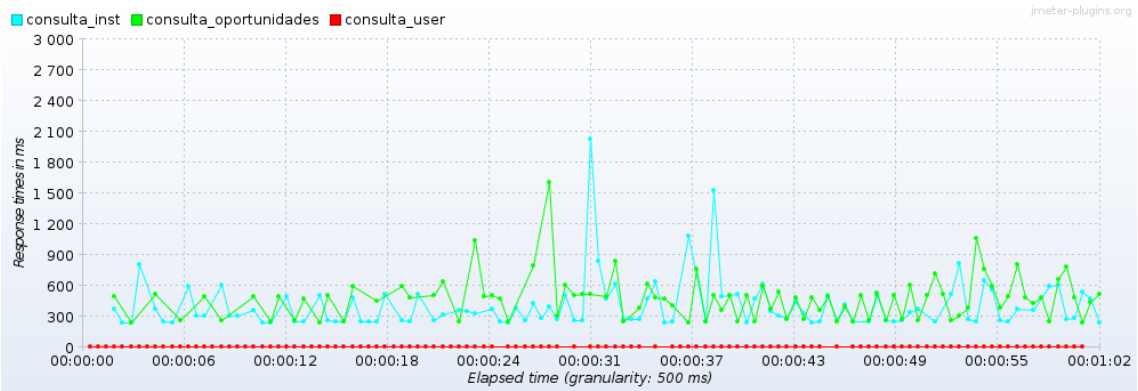

Figura 11. Tempo de resposta por consulta

A Figura 12 apresenta relação de tempo de resposta para inserção feitas pelos usuários concorrentes mostrados na Figura 10. A inserção do usuário que detêm uma rotina mais custosa de persistência, como a criação da estrutura de autenticação, tem um tempo de resposta mais elevado em relação aos outros objetos.

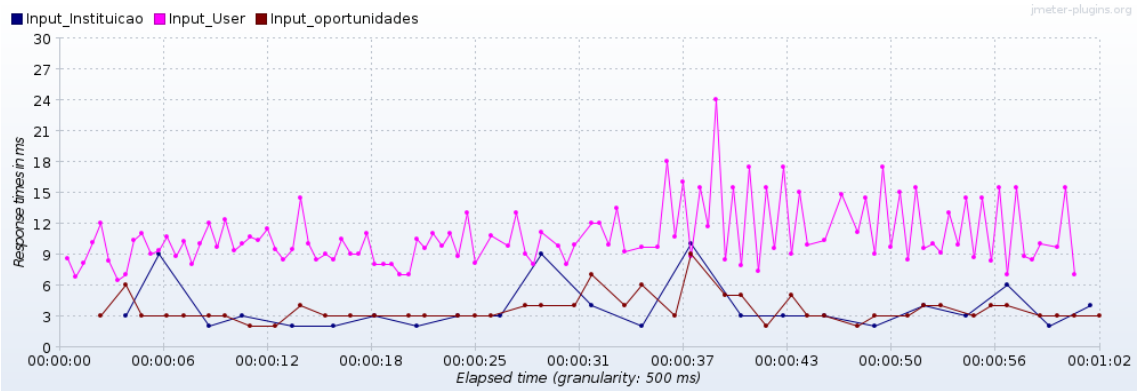

Figura 12. Tempo de resposta por inserções

Após a execução de todos os testes, o *JMeter* informa quais foram as médias de tempo de execução para cada uma das rotas nos cenários especificados (vide Tabela 2). Levando em consideração as médias exibidas pela ferramenta, observa-se que o tempo de resposta ficou dentro do padrão aceitável para o aplicativo *Ants*.

Tabela 2. Tempo médio em milissegundos de execução por minuto

Rota 1	Média(ms)	Rota 2	Média(ms)
Consulta de instituição	380,1	Cadastro de instituição	3,8
Consulta de usuário	3,7	Cadastro de usuário	9,9
Consulta de oportunidade	480	Cadastro de oportunidade	3,6

Outra execução foi feita para garantir que o aplicativo suportava 100 usuários concorrentes com tempo de resposta aceitável. Imaginado uma situação de utilização do aplicativo, a maioria dos usuários estarão realizando consultas e apenas uma parcela estarão fazendo inserções de novos usuários, oportunidades e instituições. Desta forma, simulou-se 10% dos usuários com inserções e o restante realizando consultas.

A Figura 13 mostra os usuários concorrentes adicionados a cada 30 segundos até atingir o máximo de 100 usuários. O tempo total de execução deste teste foi de 10 minutos, para garantir o desempenho e estabilidade do servidor durante o período. Este teste considerou as três histórias do usuário já apontadas previamente.

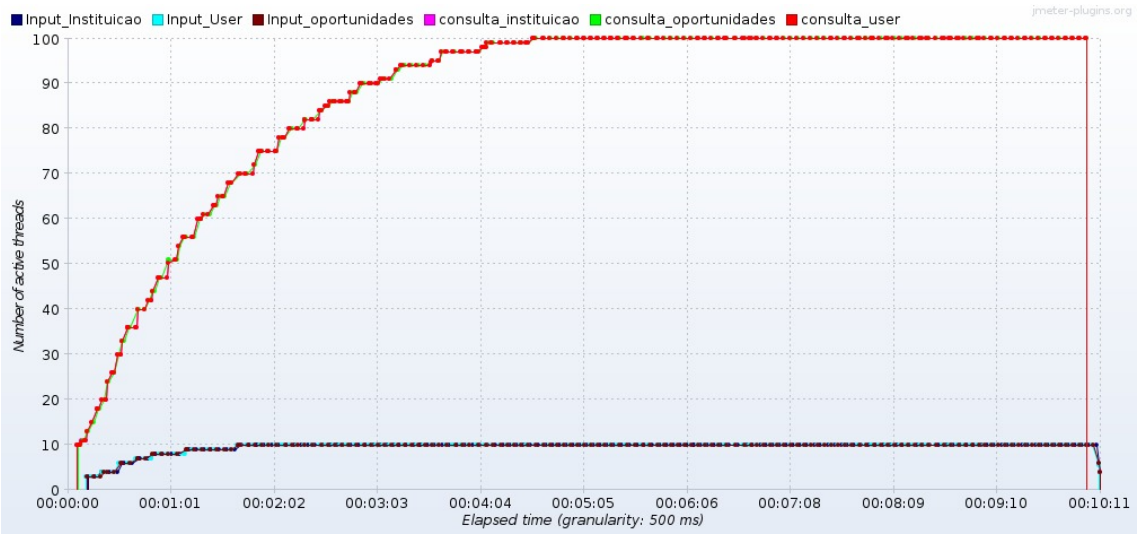

Figura 13. Número de threads por consulta e inserções durante 10 minutos de execução

A Figura 14 mostra o tempo de resposta do cenário durante os 10 minutos. Observa-se que ao atingir aproximadamente 70 usuários realizando consulta e 10 usuários realizando inserções, o tempo de resposta da consulta de instituição em média se mantém aproximadamente 1 segundo. No geral, mesmo com uma duração maior dos testes, o aplicativo manteve o tempo de resposta aceitável.

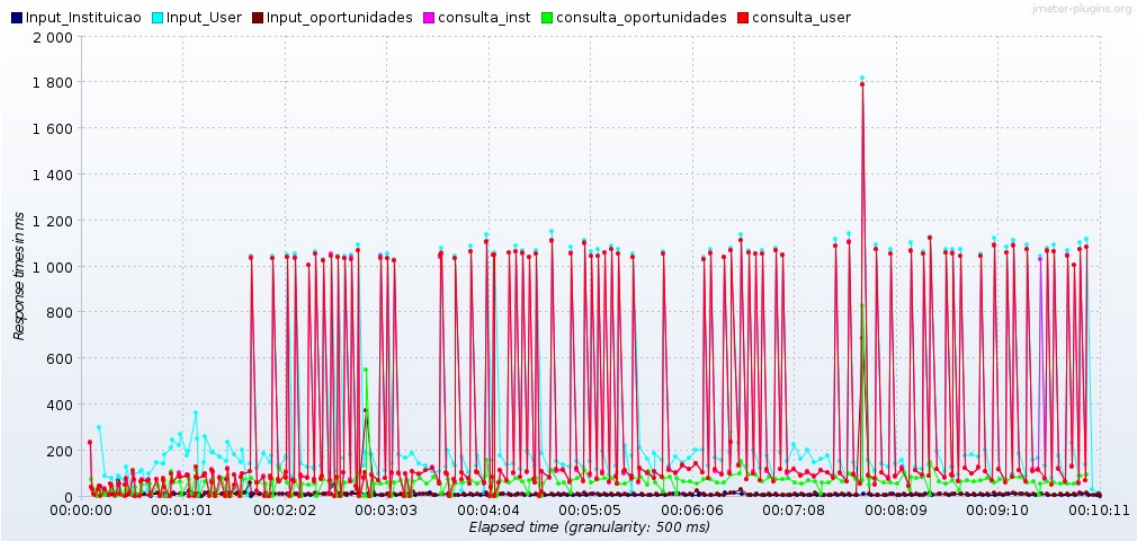

Figura 14. Tempo de resposta por consulta e inserções durante 10 minutos de execução

Com base nos resultados obtidos espera-se que, quando exposta a um ambiente produtivo, a aplicação consiga manter o baixo tempo de resposta mesmo com o advento da latência de rede e a adição de novas histórias. Observou-se que no caso dos testes de performance é necessário que todas as histórias estejam prontas para realização do teste completo e para verificar de fato o desempenho e tempo de resposta da aplicação.

5. Conclusões

Conclui-se que o objetivo de criação e execução dos testes do aplicativo *Ants* foram desenvolvidos com sucesso. Durante os estudos pode-se observar que a automação de testes auxilia bastante durante o desenvolvimento incremental, pois a cada *Sprint* desenvolvida era possível agregar novos testes referente a *Sprint* e garantir que as histórias desenvolvidas anteriormente se mantinham corretas. Em caso de problemas, era fácil detectar onde estava o problema.

A estrutura de testes disponível na ferramenta *Calabash* mostrou-se vantajosa pois foi possível gerar as estruturas após executar um comando, evitando a necessidade de criação manual.

O *JMeter* atendeu às necessidades para ser utilizado no projeto no que se refere aos testes de performance. Foi possível simular cenários, usuários concorrentes e coletar os dados de desempenho. Porém, por não termos todas as *Sprints* finalizadas, não podemos aferir que de fato o aplicativo terá um desempenho aceitável. Para que os testes de performance sejam efetivos é necessário a finalização das histórias, que o servidor esteja de fato onde será utilizado (ambiente mais próximo da produção) e que se garanta testes em vários tipos de conectividade (*Wi-fi* e 3G, por exemplo). Baseado nesses novos testes poderá ser feita a validação dos requisitos não funcionais da aplicação e de fato concluir algo com esse tipo de teste.

Esse estudo mostra que os testes funcionais foram satisfatórios para o aplicativo *Ants* e mostra que os mesmos devem se iniciar junto ao levantamento de requisitos. Testar

a aplicação e garantir sua qualidade não é apenas um processo a mais, mas sim uma das principais etapas do ciclo de desenvolvimento de um *software*, contribuindo para o sucesso do mesmo.

Finalizando, para este projeto foram utilizados conhecimentos adquiridos no curso por meio das disciplinas de algoritmos e programação, linguagem de programação, qualidade de software, engenharia de software, gestão de projetos, banco de dados, inglês técnico, entre outras disciplinas, e adquirir novos conhecimentos como testes de performance, testes funcionais, automação de testes, ferramentas *Calabash*, *Mantis*, *JMeter* e linguagem de programação Ruby. Além disso, o projeto proporcionou um trabalho de desenvolvimento de um software em equipe onde cada parte tinha suas atribuições e prazos, onde a comunicação e responsabilidade de cada um foi muito importante.

6. Referências

Apache Jmeter (2017), User's Manual. Disponível em: http://jmeter.apache.org/usermanual/index.html. Acesso em: 29 de jul. 2017.

BARTIÉ, Alexandre. Garantia da Qualidade de *software*. 5º Ed. Rio de Janeiro: Elsevier Editora Ltda. 2002.

BERNARDO, Paulo Cheque. KON, Fabio. A Importância dos Testes Automatizados. Engenharia de *software* Magazine, n 1, p. 54-57,2008. Disponível em: http://www.ime.usp.br/ kon/papers/EngSoftMagazine-IntroducaoTestes.pdf Acesso em: 08 de ago. 2017.

CAETANO, Cristiano.(2008) Melhores Práticas na Automação de Testes. "Revista Engenharia de *software* Magazine", 5a edição. p42-47.

Calabash (2017), Calabash. Disponível em: http://calaba.sh/. Acesso em: 02 de mar. 2017.

Canaltech (2017), O que é open source?. Disponível em: https://canaltech.com.br/produtos/O-que-e-open-source/. Acesso em: 29 de jul. 2017.

Capelas, B. Brasil chega a 168 milhões de smartphones em uso: Base instalada cresceu em 14 milhões de aparelhos. O Estadão, Abr. 2016. Disponível em: http://link.estadao.com.br/noticias/gadget,brasil-chega-a-168-milhoes-de-smartphones-em-uso,10000047873. Acesso em: 17 de feb. 2017.

Cucumber (2017), Cucumber. Disponível em: https://cucumber.io/. Acesso em: 29 de jul. 2017.

Mantis Bug Tracker(2017), Mantis Bug Tracker. Disponível em: https://www.mantisbt.org/. Acesso em: 29 de jul. 2017.

MATOS, Tiago et al. A utilização de testes em aplicativos móveis. Novembro.

2012. Disponivel em: http://www.tiagomatos.com/blog/a-utilizacao-de-testes-em-aplicativos-moveis. Acesso em: 13 de set. 2016.

MOLINARI, Leonardo. Testes de *software*: Produzinho Sistemas Melhores e Mais Confiáveis . 4. Ed. São Paulo: Érica, 2008.

MUCCINI, Henry et al. Software testing of mobile applications: Challenges and future research directions. 7th International Workshop on Automation of Software Test (AST), IEEE. 2012.

NETO, Arilo. Introdução a Teste de *software*. Março, 2008. Disponível em: http://www.devmedia.com.br/artigo-engenharia-de-*software*-introducao-a-teste-de-*software*/8035. Acesso em: 16 de out. 2016.

Nielsen, J. Response Times: The 3 Important Limits. Nielsen Norman Group, 2014. Disponível em: https://www.nngroup.com/articles/response-times-3-important-limits/. Acesso em: 04 de ago. 2017.

Oliveira, Rafael. Framework FUNCTEST: aplicando padrões de *software* na automação de testes funcionais. Ciência da Computação UNIFOR, Ceará. 2007. Disponível em: http://www.dominiopublico.gov.br/download/texto/cp053723.pdf. Acesso em: 01 de mar. 2017.

PEREIRA, André Luiz. O que é script?. Tecmundo, jul. 2012. Disponível em: https://www.tecmundo.com.br/programacao/1185-o-que-e-script-.htm. Acesso em: 03 de ago. 2017.

PINTO, Felipe de Oliveira Vianna. Aplicação de uma solução REST para Filantropia com uso de geolocalização. 2017. 1 v. Trabalho de Conclusão de Curso (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas), Instituto Federal de São Paulo - Câmpus Hortolândia, Hortolândia, 2017. [Orientador: Prof Msc Gustavo Bartz Guedes]

PRESSMAN, R. S., *software* Engineering: A Practitioner's Approach, McGraw-Hill, 6th ed, Nova York, NY, 2005.

PRESSMAN, Roger et al. Engenharia de *software* . 8. Ed. Porto Alegre: AMGH Editora Ltda, 2016.

PRESSMAN, Roger et al. Engenharia de *software* . 7. Ed. Porto Alegre: AMGH Editora Ltda, 2011.

Rede Brasileira do Terceiro Setor. O Terceiro Setor. Disponível em: http://www.terceirosetor.org.br/institucional/terceiro-setor. Acesso em: 14 de mar. 2017.

REGINATTO, NAISE. POR QUE AUTOMATIZAR TESTES DE SOFTWARE?.

SoftDesign, 10 abr. 2017. Disponível em: http://www.softdesign.com.br/blog/por-que-automatizar-testes-de-software/. Acesso em: 08 de ago. 2017.

Ruby (2017), Sobre o Ruby. Disponível em: https://www.ruby-lang.org/pt/about/. Acesso em: 29 de jul. 2017.

STELER, Fernando. O fim da era dos aplicativos: sua empresa está preparada para o futuro? Agosto, 2016. Disponível em: https://endeavor.org.br/como-usar-aplicativo-empresa/ Acesso em: 24 de ago. 2016.

SUTHERLAND, Jeff; SCHWABER, Ken. Cover image for *software* in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, And Leave Competitors In the Dust *software* in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, And Leave Competitors In the Dust. Nova Iorque: Wiley, 2012.

YOUNG, Michal; PEZZÈ, Mauro. Teste e análise de *software*.Porto Alegre: Bookman, 2008.