Universidade de Brasília

Faculdade UnB Gama

Engenharia de Software

APLICAÇÕES DE CONCEITO E METODOLOGIAS DE TESTES DE SOFTWARE EM UMA APLICAÇÃO REAL

Grupo 02

Gama - DF

Outubro de 2019

Autores:

Ésio G. Pereira Freitas	17/0033066	100%
Fabiana L. V. Pfeilsticker Ribas	16/0005736	100%
Fernando Ribeiro Aguilar	14/0139281	60%
Lucas Dutra F. do Nascimento	17/0050939	100%
Mikhaelle de Carvalho Bueno	15/0018673	100%
Rafael M. Gomes Ferreira	16/0142369	70%
Rogério S. dos Santos Júnior	17/0021751	100%
Youssef M. Yacoub Falaneh	17/0024334	100%

Orientador:

Dr. Ricardo Ajax Dias Kosloski

APLICAÇÕES DE CONCEITO E METODOLOGIAS DE TESTES DE SOFTWARE EM UMA APLICAÇÃO REAL

Grupo 02

Gama - DF

Outubro de 2019

RESUMO

A etapa de testes faz parte do ciclo de vida de um software e seu custo de execução é maior quando comparados com outras etapas, como o desenvolvimento e levantamento de requisitos. Isto é causado pela complexidade em realizar as técnicas e encontrar os problemas existentes. Porém, os testes são vitais para um projeto bem estruturado. Eles surgem como uma solução para diminuir o custo mais elevado ainda de manutenção e evolução do software. Fornecendo uma excelente rede de segurança que pode aumentar a confiança dos desenvolvedores na hora de adicionar recursos ou refatorar os recursos existentes.

Os testes geralmente apresentam feedbacks rápidos, podem ser de diversos tipos e voltados para diferentes campos como, por exemplo, testes unitários que buscam localizar falhas de funcionamento nas funções implementadas a nível de código no software; e, também testes de usabilidade, os quais buscam uma aceitação na parte interativa de algum usuário com as ferramentas presentes no software em questão. Dessa forma, os testes de software auxiliam em um bom desenvolvimento e entrega de um produto de software.

Este trabalho tem por finalidade, aplicar conceitos teórico e prático em um sistema real. Para que assim sejamos capazes entender como corrigir os problemas enquanto estes são pequenos, pois o custo de manutenção pode aumentar exponencialmente com o passar do tempo de projeto.

Palavras-chave: Teste de Software, Qualidade de Software, Teste de Software Ágil, Verificação e Validação de Software, Manutenção e Evolução de Software.

SUMÁRIO

INTRODUÇÃO

Teste de software é uma ótima disciplina a ser estudada no contexto do desenvolvimento de um projeto de software. Testes fazem parte do processo de elaboração de um produto, sendo capaz de reduzir a densidade de bugs em um nível considerável. Tem como objetivo capturar falhas e problemas da aplicação antes mesmos que eles cheguem no ambiente de produção, melhorando a arquitetura e a manutenibilidade do sistema.

O objetivos deste trabalho é definir o que seria uma linha base de testes para um sistema, pois a fundação de um projeto saudável é um alto padrão de testes. Quando seu software é bem testado, com testes que abrangem cenários reais e comportamentos do usuário, modificar o código não é uma tarefa tão arriscada quanto antes. É claro que isso não significa que o projeto seja à prova de balas, mas o dano potencial a seus usuários diminui.

1. OBJETIVOS GERAIS

1. Aplicar os conceitos teóricos e práticos sobre teste de software estudados na disciplina em um ambiente de desenvolvimento real,
2. Estudar e documentar a parte teórica sobre os tipos de testes e metodologias para aplicá-los de forma prática em um Software em desenvolvimento para assim encontrar pontos fortes, detectar defeitos e falhas e propor soluções de melhoria levando em conta as métricas colhidas e resultado dos testes,
3. Praticar a abordagem *Goal Question Metrics* (GQM) para encontrar as métricas corretas que devem ser aplicadas ao software em estudo e colher os dados de medição para utilizá-los em tomadas de decisão e aperfeiçoamento futuros

2. OBJETIVOS ESPECÍFICOS

- Detectar possíveis defeitos e falhas;
- Definir métricas que possam abranger aspectos importantes que devem ser aplicadas ao software em estudo;
- Verificar a funcionalidade dos componentes;
- Verificar a integração dos componentes;
- Validar a usabilidade do sistema;
- Colher dados de medição;
- Propor melhorias futuras;
- Projetar casos de teste.

3. TEMA

Como tema, a equipe optou por utilizar um software já conhecido pela maioria dos membros. Este software se chama PAX e foi desenvolvido por boa parte dos membros em uma disciplina do curso de Engenharia de Software da Universidade de Brasília - UnB. A metodologia usada para o desenvolvimento do software foi o Scrum, Kanban e XP e utiliza das tecnologias *Flutter* para *Frontend*, *Flask* e *Express* para o *Backend, PostgreSQL* e

Firebase para o banco de dados e utiliza de Micro Serviços como abordagem arquitetural. O software ainda está na fase de desenvolvimento e os testes selecionados podem ser aplicados.

Serão realizados testes unitários, de integração e usabilidade aplicados no sistema em produção, nas partes já desenvolvidas a partir da brach mais estável do sistema no ambiente de produção, com o propósito de ter maior contato com a dinâmica da vida real. Dessa forma, o software que a equipe testará neste documento estará sendo guiado por métodos, técnicas e ferramentas apresentadas em sala de aula pelo professor Ajax tais como, métricas internas e externas, abordagem GQM e aplicação de teste de software.

4. PROBLEMA

4.1 Aplicação testada

Pax é um aplicativo móvel multiplataforma que tem como princípio facilitar a contratação de prestadores de serviços, estabelecendo uma conexão direta entre o consumidor e o prestador de serviços. O aplicativo estabelece uma relação de harmonia entre os dois tipos de usuários, o consumidor obtém uma maior praticidade ao encontrar bons prestadores de serviços, e os prestadores ganham mais reconhecimento. O aplicativo entrega a proposta de facilitar a comunicação entre as partes através de um serviço de chat além de um módulo de pagamento via app.

A aplicação foi elaborada na matéria de Desenho e Arquitetura de Software com o intuito de praticar os conceitos vistos em sala de aula, tais como levantamento de requisitos, padrões de projeto e arquitetura de software . A documentação do projeto pode ser acessada no repositório do projeto[3].

4.3 Motivos dos testes

Por ser um software desenvolvido do zero, está em sua forma de mínimo produto viável e não possui ciclos de manutenção, os testes tem a finalidade de verificar, validar e garantir a qualidade e eficiência do software para o usuário final.

Primeiro, é necessário escrever testes para obter alguns dos benefícios listados: melhorias na arquitetura, melhor design da experiência do desenvolvedor e feedback mais rápido à medida que você desenvolve o sistema.

Com relação à qualidade interna do produto, os testes podem indicar valores de desempenho do software quando aplicadas às funções implementadas em nível de código, no caso dos testes unitários. Podem, também, indicar a interação do produto com diferentes interfaces por meio dos testes de mobilidade.

Além disso, os testes garantem a qualidade do projeto, ou seja, proporciona indicadores que afirmam a qualidade do desenvolvimento e do produto como um todo. Dessa forma, há uma maior facilidade na análise, evolução e refatoração de quaisquer visão do projeto.

Os testes fornecem parâmetros mensuráveis que ajudam a aumentar a confiança do time de desenvolvimento na hora de adicionar recursos novos ou refatorar os recursos existentes.

4. AMBIENTE DE TESTES

O desenvolvimento dos testes a serem aplicados, visando a padronização e diminuição de possíveis erros de codificação, será feita com o apoio das seguintes ferramentas:

- Visual Studio Code: ferramenta integrada de desenvolvimento robusta, o qual é de maior familiaridade pelos membros do grupo, que auxilia na programação por meio de vários plugins fornecidos pela comunidade;
- *Prettier* + *ESLint* (*JavaScript*) + *PyLint* (*Python*): plugins que formatam o código ao simples comando de salvar e fazem uma varredura para encontrar erros de sintaxe. Isso proporciona maior uniformidade aos códigos desenvolvidos e menor probabilidade de inserção de erros;

Para o desenvolvimento dos testes foram escolhidas algumas dependências, de acordo com a linguagem do sistema, para poder definir os teste unitários, sendo elas:

- *UnitTest* (*Python*);
- *Mocha* / *Chai* (*JavaScript*);
- *FlutterTest* (*Flutter*);

Para o teste de integração, dada a arquitetura do sistema e a linguagem usada, foi escolhida a dependência *SuperTest* para o *JavaScript*, uma vez que os teste serão realizados sobre a *Gateway*, serviço responsável pela comunicação dos microsserviços, por ser justamente o responsável por mediar os subsistemas do Sistema.

Para analisar os sistemas, baseados nos testes citados anteriormente, foram escolhidas as seguintes ferramentas:

- Codeclimate: ferramenta integrada ao repositório do GitHub que analisa os códigos da branch de desenvolvimento de forma a verificar estaticamente a qualidade de código. Encontra *code smells*, duplicações e outros problema que podem haver no código, além de fornecer algumas métricas que podem auxiliar no planejamento da execução dos testes;
- Codecov: ferramenta usada para analisar a cobertura do sistema em relação aos testes unitários;

Depois de algumas pesquisas, para os testes de usabilidade, foram escolhidas as ferramentas abaixo, que se mostraram pertinentes ao contexto que se deseja atingir:

- HeatMap: técnica que mostra aos designers quais foram os lugares de maior interação de cliques com o usuário;
- Usability Hub: plataforma que proporciona ferramentas para medir a interação do usuário com a aplicação mobile de forma a realizar questionários baseados nas telas do aplicativo. Dependendo do foco do teste, a ferramenta fornece várias métricas a partir da interação dos usuários.

5. EXECUÇÃO

Os testes serão executados seguindo os cenários mapeados e os critérios das histórias estabelecidas. Além disso, também serão executados nas implementações que não estão funcionando corretamente. No fim, os resultados devem ser informados à equipe de desenvolvimento, para que possam efetuar as devidas correções e voltar para a etapa de testes novamente.

6. MÉTODO

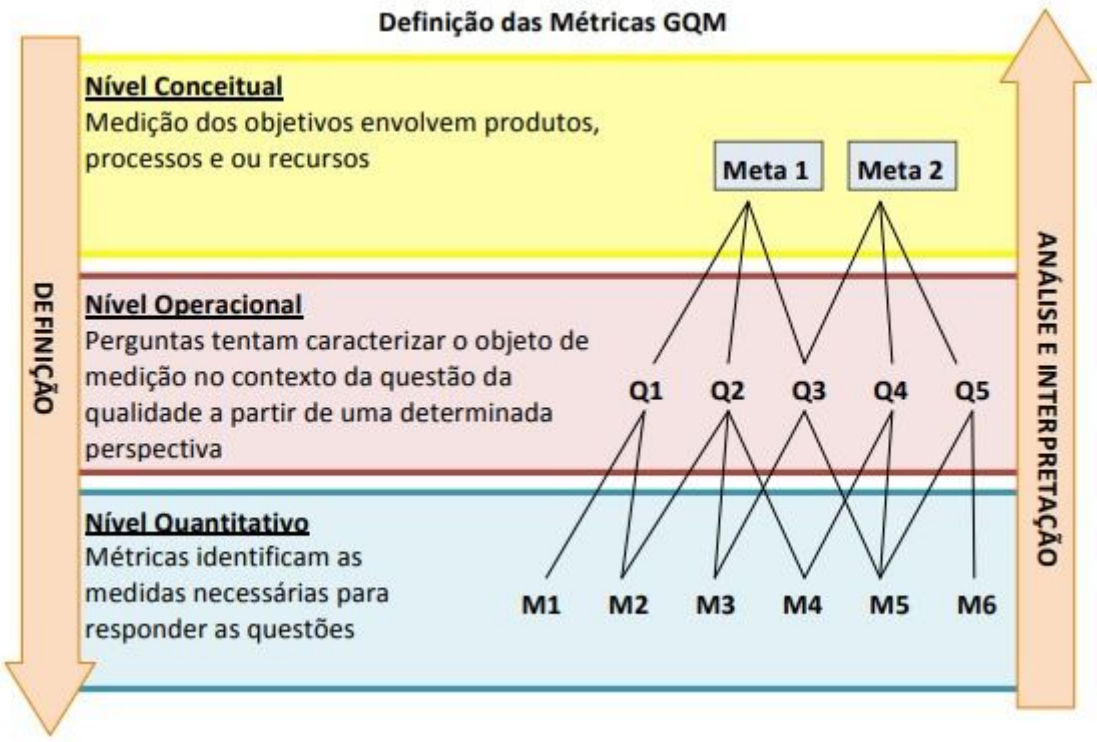
Por se tratar de um trabalho em desenvolvimento, a equipe decidiu abordar a metodologia de testes ágeis. A aplicação de testes é realizada em todas as etapas do processo de desenvolvimento. No entanto, haverá uma equipe especializada na realização de testes, visto que os times são diferentes. Assim a equipe de testes pode trabalhar de forma mais independente. Para a organização dos teste será adotada a metodologia GQM ou Goal Question Metrics.

Desenvolvida pelo Dr.Victor Basili e colegas durante a década de 1980, em conjunto com o seu trabalho no Laboratório de Engenharia de Software da NASA (SEL). O GQM é um modelo que é bastante usado para definir, implantar, medir, analisar e melhorar os processos. O GQM consiste em um processo para identificar os objetivos para os quais buscamos respostas e a partir deles definir questões que permitam a definição de métricas para se realizar a medição. Os resultados da medição servem para guiar a tomada de decisão e futuros aperfeiçoamentos.

Segundo Basili descreve seis passos do processo do GQM do seguinte modo:

“[...

1. Descrever um conjunto de metas de negócio e suas medições com o objetivo de conseguir produtividade e qualidade;
2. Gerar perguntas que definem essas metas o tanto quanto possível de um modo quantitativo;
3. Especificar as medidas que necessitam ser coletadas para responder essas perguntas e acompanhar o processo e a conformidade do produto em relação às metas;
4. Desenvolver mecanismos de coleta de dados;
5. Coletar, validar e analisar os dados em tempo real para providenciar feedback para os projetos, com o objetivo de tomar ações corretivas nos mesmos;
6. Analisar os dados em um formato de post mortem para avaliar a conformidade com as metas e criar recomendações para futuras melhorias...]”[5]



[O modelo funciona da seguinte forma: 1- Abordagem de cima para baixo (top-down); 2 - Direcionado a metas; 3 - Pode ser aplicado a todo o ciclo de vida de produtos, processos e recursos;]

Alguns passos são necessários para atingir o objetivo deste trabalho, que é testar uma aplicação mobile desenvolvida na disciplina de Desenho e Arquitetura de software. A primeira etapa consiste em definir o escopo que se deve testar. O projeto de software criado está organizado dentro de um repositório contendo diversos serviços, sendo um deles o principal por conter toda a documentação do projeto.

O GQM consiste em identificar os objetivos para os quais buscamos respostas e a partir deles definir questões que permitam a definição de métricas para se realizar a medição, neste caso, dentro do projeto testado. Por fim, os dados serão estudados para que haja as devidas correções e evoluções em cima do software criado.

7. TIPOS DE TESTES

Os tipos de testes a serem descritos serão os utilizados pela equipe no desenvolvimento do trabalho. Foram escolhidos baseados na necessidade e adequação no contexto da aplicação escolhida.

Tipo de Teste	Descrição
Integração	O Teste de Integração é um nível de teste de software em que unidades individuais são combinadas e testadas como um grupo
Unitário	Teste unitário é um nível de teste de software no qual unidades / componentes individuais de um software são testados.
Usabilidade	O teste de usabilidade é uma maneira de ver o quão é fácil utilizar algo testando-o com usuários reais.

Tabela 1 - Tipos de Testes e Suas Respectivas Descrições.

8. METODOLOGIA DE APLICAÇÃO DOS TESTES

8.1 Teste de Integração

O objetivo do teste de integração, como o nome sugere, é testar se muitos módulos desenvolvidos separadamente funcionam juntos conforme o esperado. Isso foi realizado ativando muitos módulos e executando testes de nível superior contra todos eles para garantir que eles operassem juntos. Esses módulos podem partes de um único executável ou separado. Definições dadas pelo ISTQB (International Software Testing Qualifications Board):

- Teste de integração: Teste realizado para expor defeitos nas interfaces e nas interações entre componentes ou sistemas integrados.
- Teste de integração de componentes: Teste realizado para expor defeitos nas interfaces e interação entre componentes integrados.
- Testes de integração de sistemas: Testando a integração de sistemas e pacotes; teste de interfaces com organizações externas (por exemplo, Electronic Data Interchange, Internet).

Por utilizar a abordagem de arquitetura de microsserviços, o teste de integração é de suma importância para o planejamento de testes da equipe, pois o mesmo irá assegurar a consistência e confiabilidade das interações entre os serviços, validando as requisições, as regras de de negócios e as respostas.

Como na arquitetura do software em desenvolvimento existe um módulo responsável por coordenar a comunicação do *front-end* com os demais microsserviços, a *API Gateway*, é essencial que este módulo esteja devidamente testado já que ele é o principal ponto de falha. O teste de integração neste cenário visa realizar os principais fluxos de comunicações entre os serviços da aplicação para que exista a segurança entre as trocas de dados, e verificar a estabilidade da aplicação caso surjam novas atualizações.

8.1.1 Template GQM para definição do objetivo do Teste de Integração

Análise	Integração dos serviços
Propósito	Garantir a comunicação uniforme dos sistemas
Com respeito	A confiabilidade
Sob o ponto de vista	Dos desenvolvedores
No contexto	Da disciplina de Testes de Software

8.1.2 Abstraction Sheet GQM

Objeto	**Propósito**	**Fator de qualidade**	**Ponto de vista**
Produto de Software	Garantir a comunicação uniforme dos micro serviços do sistemas	Confiabilidade	Desenvolvedores

8.1.3 Questões

Q1	Os microsserviços estão se comunicando corretamente?
Q2	Os microsserviços estão transmitindo e recebendo os dados da forma esperadas?

8.1.4 Métricas

M1	Informações transmitidas e recebidas corretamente
M2	Informações transmitidas corretamente mas recebidas incorretamente
M3	Resultado da interação produzido corretamente

8.1.5 Definições das métricas

M1: A comunicação está acontecendo corretamente

M2: Alguma informação teve interferência ou se perdeu durante a comunicação

M3: A interação entre os microsserviços teve o output esperado

8.1.6 Dependência

Q1	M1, M2
Q2	M1, M2, M3

8.1.7 Medição

Uma forma de medir a taxa de comunicações executadas bem sucedidas, com foco na confiabilidade, é dada pela seguinte equação, sendo n a quantidade de testes e T o teste executado:

$$Confiabilidade = \frac{\sum_{n}^{1} T_{sucesso}}{n} \times 100$$

8. 2 Teste Unitário

O objetivo dos testes unitários é validar que cada unidade do software funcione conforme projetado. Uma unidade é a menor parte testável de qualquer software. Geralmente possui uma ou algumas entradas e geralmente uma única saída. *Frameworks* de testes unitários, *drivers*, *stubs* e *mocks* são usados para ajudar no teste unitário.

A definição de unidade gera algumas diferenças na aplicação dos testes unitários. O design orientado a objetos tende a tratar uma classe, pois a unidade, as abordagens processuais ou funcionais podem considerar uma única função como unidade. Mas realmente é uma coisa situacional - a equipe decide o que faz sentido ser uma unidade para os propósitos de sua compreensão do sistema e seus testes.

A equipe fará uso do teste unitário, principalmente, para testar comportamentos de *views*/*controllers* implementadas nos serviços da aplicação, validando a comunicação primordialmente com o banco de dados.

8.2.1 Template GQM para definição do objetivo do Teste de Integração

Análise	Do código
Propósito	Garantir a integridade dos métodos
Com respeito	Confiabilidade
Sob o ponto de vista	Produto
No contexto	Disciplina de Testes de Software

8.2.2 Abstraction Sheet GQM

Objeto	Propósito	Fator de qualidade	Ponto de vista
Produto de Software	Garantir a integridade dos métodos do sistema	Confiabilidade	Produto

8.2.3 Questões

Q1	A função cobre todo o escopo de resolução que está inserida?
Q2	A função executa corretamente o que foi proposto?

8.2.4 Métricas

M1	função executa corretamente
M2	função trata casos de exceção
M3	função cobre os casos que estão dentro do seu contexto

8.2.5 Definições das métricas

M1: O resultado pós-processamento da função está coerente com o proposto

M2: Inputs indevidos do usuário são tratados

M3: A função é completa dentro do seu contexto de atuação

8.2.6 Dependência

Q1	M3
Q2	M1, M2, M3

8.2.7 Medição

A forma de aferir a confiabilidade das métricas propostas se dá pelo uso da seguinte equação, sendo *n* a quantidade de testes e *T* o teste executado:

$$Confiabilidade = \frac{\sum_{n}^{1} T_{sucesso}}{n} \times 100$$

8. 3 Teste de Usabilidade

Testes de usabilidade é um método utilizado que pode abranger diversos focos, podendo ser desde o quão bonita uma aplicação está, até o acessibilidade da mesma. Esse tipo de teste é realizado com usuários reais para obter-se métricas passíveis de ação a partir dos resultados obtidos dos cenários pré-definidos.

A principal diferença entre o teste de usabilidade e os testes tradicionais (testes de aceitação, testes unitários, etc.) é que ele testa com usuários reais ou clientes do produto. Enquanto testes tradicionais são provavelmente executados por um desenvolvedor, designer ou gerente do projeto, o teste de usabilidade remove qualquer tipo de enviesamento e coleta *feedback* diretamente com o usuário.

Os mais relevantes objetivos e insumos coletados por essa técnica são:

- Adquirir opiniões sobre o projeto diretamente com o usuário;
- Verificar se o que foi desenvolvido corresponde às expectativas do usuário;
- Verificar se o design da aplicação corresponde às decisões de negócio tomadas para o uso no mundo real;
- Verificar se o usuário consegue fazer as tarefas propostas;
- Descobrir se o desenvolvimento está no caminho certo para aceitação do público alvo;
- Obter reações e *feedbacks* dos usuários.

Por se tratar de uma aplicação *mobile* com um público alvo bastante abrangente, a equipe de desenvolvimento optou por executar, também, testes de usabilidade com o intuito principal de validar suas funcionalidades, colhendo os resultados para pontuar o artefato S.I.G.(Softgoal Interdependency Graph) validando tanto o artefato, quanto o software, para que seja possível tomar iniciativas para mudanças com base em um referencial validado.

Softgoal Interdependency Graph, mais conhecido como S.I.G. ou N.F.R.(Non Functional Requirements), é uma representação gráfica para modelar requisitos não funcionais(*softgoal*), decompondo-os em outros requisitos não funcionais até chegar em requisitos funcionais, podendo haver interferências destes requisitos decompostos nos demais *softgoals*. Após a parte de decomposição é necessário validar todos os requisitos com os clientes para que seja possível pontuar todos os requisitos presentes no modelo.

8.3.1 Template GQM para definição do objetivo do Teste de Integração

Análise	Da interface do aplicativo
Propósito	Validar a aplicação, na visão do usuário
Com respeito	Eficiência
Sob o ponto de vista	Usuário
No contexto	da matéria de Testes de software

8.3.2 Abstraction Sheet GQM

Objeto	Propósito	Fator de qualidade	Ponto de vista
Interface da aplicação	Avaliar se o usuário consegue concluir com sucesso uma determinada tarefa	Eficácia	Usuário

8.3.3 Questões

Q1	Os usuários conseguiram completar o fluxo do sistema sem dificuldade?
Q2	Os usuários completaram as tarefas de moda a realizar os objetivos propostos?

8.3.4 Métricas

M1	Conclusão de tarefas sem erro
M2	Conclusão de tarefa com erro (não crítico)

M3	Erros críticos

8.3.5 Definições das métricas

M1: Quantidade de usuários que conseguem concluir com sucesso a tarefa sem cometer nenhum erro;

M2: Um erro não crítico é um erro que não teria impacto no sucesso da tarefa, mas resultaria na tarefa ser concluída com menos eficiência;

M3: São erros que não são resolvidos na tentativa de realização da tarefa e que impedem a pessoa de finalizar e a desistir de continuar. Podemos considerar como erro crítico também todas a vezes que o usuário não consegue prosseguir sozinho e solicita ajuda para prosseguir no fluxo;

8.3.6 Dependência

Q1	M1,M2,M3
Q2	M1,M2,M3

8.3.7 Medição

Uma forma utilizada para medir a taxa de conclusão de tarefa com foco em eficácia é a seguinte, sendo n a quantidade de testes e T o teste executado:

$$Ef\,icá\,cia = \frac{\sum_{n}^{1} T_{sucesso}}{n} \times 100$$

REFERÊNCIAS

Pax-App. Disponivel em: https://pax-app.github.io/Wiki/#/. Acesso em: 20 de Outubro de 2019.

Vinícius. PROCESSO DE TESTE ÁGIL X TRADICIONAL . DEVMEDIA. 2016. Disponível em: www.devmedia.com.br/processo-de-teste-agil-x-tradicional/36854. Acesso em: 13 de mai. 2019.

HOW TO TEST MOBILE APPLICATION. Disponível em: https://geteasyqa.com/qa/mobile-apps-testing. Acesso em: 25 de mai. 2019.

Pressman, Roger S. Engenharia de Software: Uma abordagem profissional . 8o edição, 2016.

What is usability testing. Disponível em: https://www.experienceux.co.uk/faqs/what-is-usability-testing/. Acesso em: 20 de out. de 2019

V.R. Basili,"Software Modeling and Measurement: The Goal Question Metric Paradigm,Página 4

Usability testing: what is it and how to do it? Disponível em: https://uxdesign.cc/usability-testing-what-is-it-how-to-do-it-51356e5de5d. Acesso em: 20 de out. de 2019

Unit Test. Disponível em: https://martinfowler.com/bliki/UnitTest.html. Acesso em: 20 de out. de 2019

Unit Testing. Disponível em: http://softwaretestingfundamentals.com/unit-testing/. Acesso em: 20 de out. de 2019

Integration Testing. Disponível em: http://softwaretestingfundamentals.com/integration-testing/. Acesso em: 20 de out. de 2019

Integration Test. Disponível em: https://martinfowler.com/bliki/IntegrationTest.html. Acesso em: 20 de out. de 2019