Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Testes regressivos visuais: uma aplicação em um projeto Android

Lucas Apolinário Figueiredo

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Igor Muzetti Pereira

COORIENTAÇÃO:
Vicente José Peixoto de Amorim

Fevereiro, 2018
João Monlevade–MG

Lucas Apolinário Figueiredo

Testes regressivos visuais: uma aplicação em um projeto Android

Orientador: Igor Muzetti Pereira

Coorientador: Vicente José Peixoto de Amorim

Monografia apresentada ao curso de Sistemas de Informação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Fevereiro de 2018

FOLHA DE APROVAÇÃO DA BANCA EXAMINADORA

Testes regressivos visuais: uma aplicação em um projeto Android

Lucas Apolinário Figueiredo

Monografia apresentada ao Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto como requisito parcial da disciplina CSI499 – Trabalho de Conclusão de Curso II do curso de Bacharelado em Sistemas de Informação e aprovada pela Banca Examinadora abaixo assinada:

Igor Muzetti Pereira

Prof. Igor Muzetti Pereira
Mestre em Ciência da Computação
DECSI – UFOP

Vicente José Peixoto de Amorim
Mestre em Ciência da Computação
DECSI – UFOP

Darlan Nunes de Brito
Mestre em Modelagem Matemática e Computacional
Examinador
DECSI – UFOP

Elton M. Cardoso

Elton Máximo Cardoso
Mestre em Ciência da Computação
Examinador
DECSI – UFOP

João Monlevade, 21 de fevereiro de 2018

ATA DE DEFESA

No dia 21 do mês de Fevereiro de 2018, às 15:00 horas, na sala H102 do Instituto de Ciências Exatas e Aplicadas, foi realizada a defesa de Monografia pelo(a) aluno(a) **Lucas Apolinário Figueiredo**, sendo a Comissão Examinadora constituída pelos professores: Prof. Igor Muzetti Pereira, Prof. Vicente José Peixoto de Amorim , Prof. Darlan Nunes de Brito, Prof. Elton Máximo Cardoso. O(a) candidato(a) apresentou a monografia intitulada: "**Testes regressivos visuais: uma aplicação em um projeto Android**". A comissão examinadora deliberou, por unanimidade, pela aprovação do candidato, com nota 9.0 (oito), concedendo-lhe o prazo de 15 dias para incorporação das alterações sugeridas ao texto final. Na forma regulamentar, foi lavrada a presente ata que é assinada pelos membros da Comissão Examinadora e pelo(a) graduando(a).

Igor Muzetti Pereira
Prof. Igor Muzetti Pereira
Mestre em Ciência da Computação
DECSI – UFOP

Vicente José Peixoto de Amorim
Mestre em Ciência da Computação
DECSI – UFOP

Darlan Nunes de Brito
Mestre em Modelagem Matemática e Computacional
Examinador(a)
DECSI – UFOP

Elton M. Cardoso
Elton Máximo Cardoso
Mestre em Ciência da Computação
Examinador(a)
DECSI – UFOP

Lucas Apolinário Figueiredo
Lucas Apolinário Figueiredo

João Monlevade, 21 de fevereiro de 2018

TERMO DE RESPONSABILIDADE

Eu, **Lucas Apolinário Figueiredo** declaro que o texto do trabalho de conclusão de curso intitulado "***Testes regressivos visuais: uma aplicação em um projeto Android***" é de minha inteira responsabilidade e que não há utilização de texto, material fotográfico, código fonte de programa ou qualquer outro material pertencente a terceiros sem as devidas referências ou consentimento dos respectivos autores.

João Monlevade, 21 de fevereiro de 2018

Lucas Apolinário Figueiredo

Este trabalho é dedicado especialmente ao meu pai Gisnaldo, minha mãe Lutiane, minha irmã Júlia, minha esposa Laila e ao meu filho Heitor. Dedico também à minha família, amigos e todos os envolvidos.

Agradecimentos

Agradeço primeiramente a Deus por ter me dado saúde e força para enfrentar todos os obstáculos que enfrentei. Agradeço também aos meus pais e ao meu sogro Ivan, que deram todo o incentivo para que eu desse continuidade aos meus estudos. Que sempre me apoiaram e me deram forças durante toda minha trajetória de vida. Agradeço a minha esposa e ao meu filho que foram as pessoas que fizeram com que eu me dedicasse acima da média e alcançasse a graduação antes do esperado. Agradeço também aos meus professores que me guiaram durante a trajetória acadêmica. Aos meus colegas e amigos do iMobilis que tornaram a trajetória mais divertida e animada durante os *coffee times*, e as jogatinas de *shit happens* e ao meu orientador que me guiou durante as pesquisas.

"Science is more than a body of knowledge; it is a way of thinking."

— Carl Sagan (1934 – 1996),
in: The Demon-Haunted World: Science as a Candle in the Dark.

Resumo

O desenvolvimento de *software* sem um processo de testes pode ocasionar em produtos finalizados com muitas falhas, facilmente detectadas pelos usuários durante seu uso. O processo de testes visa, através de coleta de informações essenciais sobre o sistema e a forma como o usuário o manipula, estabelecer um ciclo de atividades, verificando o funcionamento das funcionalidades e validando os seus cenários de testes. Este trabalho descreve como foi um processo de testes ágil que se adaptou a um projeto de desenvolvimento de uma aplicação Android em um contexto de pesquisa e inovação. Esta abordagem ágil permitiu a automação de testes de aceitação e regressão em conjunto com testes exploratórios. A proposta desse trabalho consiste na utilização de comparação de imagens dentro de um processo de teste, com o objetivo de comparar estados válidos através de imagens da aplicação após uma sequência de ações. O processo proposto trouxe um ganho positivo em relação à qualidade agregada ao produto e o tempo gasto pelo analista de testes na execução dos mesmos.

Palavras-chaves: testes de *software*. processo de teste. automação de testes. testes exploratórios. testes regressivos. comparação de imagens.

Abstract

Software development without a testing process can result in finalized products with many flaws that could be easily detected by user interaction. A testing process aims to create a cycle of activities by collecting essential information about the system and how the user uses it, verifying the functionalities and validating tests scenarios. This work describes an agile test process that has been adapted to an Android project in a context of research and innovation. This agile approach allowed the automation of acceptance and regression tests with exploratory tests. The main purpose of this work is to compare valid states through images of an application after a sequence of actions. The proposed process has a very positive gain in quality added to the project and in time spent by the analyst during tests execution.

Key-words: software testing. test process. automation tests. exploratory tests. regressive tests. images comparison.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ICEA Instituto de Ciências Exatas e Aplicadas

UFOP Universidade Federal de Ouro Preto

PO *Product Owner*

Sumário

1 Introdução

O processo de teste em desenvolvimento de *software* é algo de suma importância, pois, através dele, juntamente com algumas outras práticas oriundas da qualidade de *software*, podemos garantir uma maior qualidade de um produto (SOMMERVILLE, 2011). Quando práticas de testes ágeis são aplicadas, as necessidades dos usuários que utilizarão o *software* se tornam prioritárias. Ou seja, o foco da equipe de desenvolvimento e testes é entregar um produto com boa usabilidade e confiabilidade (CRISPIN; GREGORY, 2009)

Testes de Aceitação, é uma categoria que tem como finalidade a verificação de um produto de *software* em relação aos requisitos e necessidades do usuário (DUSTIN; RASHKA; PAUL, 1999). Ele é comumente usados em equipes ágeis, por serem de fácil planejamento, e ser direcionado a maneira como o usuário manipula o *software*. Contudo, a prática de quaisquer testes de maneira manual é um tanto quanto dispendiosa e cara. Felizmente, algumas categorias podem ser automatizadas, sendo os testes de aceitação e regressão algumas delas. A automação de testes além de ser uma prática onde se executa um conjunto de testes de maneira rápida e independente, promove maior eficiência por executar repetitivamente um mesmo conjunto de testes. Em especial, é uma prática muito recomendada nos testes regressivos, onde esses são executados de maneira incremental e iterativa devido às mudanças que normalmente ocorrem em projetos reais de *software* (DUSTIN; RASHKA; PAUL, 1999).

Laboratórios acadêmicos de pesquisa buscam o desenvolvimento de ideias, promovem pesquisas voltadas a uma área um pouco mais específica dos cursos de graduação e pós-graduação. O contato de graduandos com a pesquisa desperta o interesse em identificar problemas e possíveis formas de os solucionarem, relacionando os ensinamentos de sala de aula com a prática, dentre outras habilidades e conhecimentos que lhe acompanharão ao longo da vida. Esse ambiente, além de proporcionar um contato maior do graduando com o meio acadêmico, também o aproxima de um ambiente profissional da indústria (RODRIGUES; ESTRELA, 2012). Com propósito semelhante, o laboratório de computação móvel iMobilis[1], situado no Instituto de Ciências Exatas e Aplicadas (ICEA), campus da Universidade Federal de Ouro Preto (UFOP), vem ao longo do tempo implantando práticas utilizadas atualmente no mercado, dentre elas podem-se citar metodologias híbridas de gerenciamento de projetos, no caso do laboratório o BOPE *process*, (PEREIRA; CARNEIRO; PEREIRA, 2013) e Integração Contínua (NUNES, 2017). Tais métodos fazem com que os alunos de graduação que participam do laboratório tenham contato prático com conceitos vistos em sala de aula, fazendo com que seja entregue ao final dos projetos, produtos de *softwares* desenvolvidos com os requisitos atendidos e dentro do

[1] <https://www.imobilis.ufop.br/>

prazo estimado, resultando no sucesso dos projetos (REZENDE, 2005).

Dentre os projetos desenvolvidos pelo iMobilis, pode-se destacar o MobMine. O projeto é desenvolvido em parceria com uma organização privada e consiste em uma solução móvel (Android) para o desenvolvimento de uma aplicação que otimize um de seus processos internos. Esse projeto é composto por um aplicativo voltado para *tablets*, uma ferramenta de desenho 2D incorporada a esse aplicativo e um servidor web. Este projeto tem duração prevista de dois anos, sua equipe é composta por dez membros, além de um cliente que representa a empresa e tem papel de *Product Owner* (PO). Dentre os membros, três são professores mestres, um graduado é pesquisador e líder da equipe, um é analista de testes e os demais são desenvolvedores.

A automação de testes de aceitação e de regressão foi realizada visando garantir a qualidade do produto de *software* desenvolvido e encontrar a maior ocorrência de falhas na fase de desenvolvimento. Desta forma foi direcionado a aplicação móvel, pois a mesma apresenta uma alta interação com o usuário, uma alta complexidade e uma grande quantidade de requisitos, ao contrário do servidor web.

Os testes realizados sobre os elementos gráficos nativos do Android ocorreram de maneira natural, por meio do uso da ferramenta Robotium[2]. Ela facilita a manipulação de elementos gráficos do Android utilizando o JUnit[3], tornando a codificação dos testes de aceitação mais simples e ágil. Porém, os testes realizados sobre os elementos gráficos que foram gerados pela ferramenta de desenho 2D, ocorriam de maneira manual por diversos fatores. Tais fatores são: o Robotium não consegue identificar os elementos visuais gráficos gerados pela ferramenta de desenho; a captura de referência dos elementos gerados pela ferramenta de desenho via código, é um tanto quanto complexa, dispendiosa e partiria do pressuposto que o código não apresenta defeitos, algo não trivial; e por fim, não seria possível fazer uma averiguação dos cenários de testes por meios do JUnit de maneira rápida e eficiente, pois levaria o analista de teste despender muito tempo para projetar e desenvolver um único cenário de teste. Atrasando todo o processo de verificação e validação de uma funcionalidade comprometendo a agilidade do processo e do cronograma do projeto.

Em busca da melhor eficiência do processo de testes por meio da validação rápida e correta, do desenvolvimento simples e ágil de código de testes e da execução dos testes de maneira automatizada, este trabalho tem como objetivo principal a construção de um processo de validação dos cenários de testes. Cenários esses decorrentes das funcionalidades apresentadas pela ferramenta de desenho 2D, por meio de comparação de imagens. Imagens essas, que serão geradas através de capturas de telas durante o processo de automação dos testes e serão comparadas com imagens que representam o resultado desejado, após a mesma sequência de ações. Tudo isso integrado na execução do próprio cenário de teste.

[2] <https://github.com/RobotiumTech/robotium>
[3] <http://junit.org/junit4/>

O objetivo secundário deste estudo foi alcançar a qualidade do produto de *software* decorrente do projeto, através de um processo de teste. Visando essa finalidade, o processo de testes foi composto de: testes automatizados de aceitação e de regressão, testes manuais complementares e exploratórios e uma abordagem híbrida de automação de testes exploratórios de maneira supervisionada, além da validação dos cenários de testes por meio da comparação de imagens.

O restante deste trabalho é organizado como se segue. O Capítulo 2 apresenta os conceitos teóricos abordados e os trabalhos correlatos. No Capítulo 3 são apresentadas as metodologias do trabalho, o processo de testes utilizado e a forma como foi organizada e executada a comparação de imagens dentro do processo de teste. Os resultados iniciais serão apresentados no Capítulo 4 e no último capítulo, o Capítulo 5, serão apresentadas as conclusões, desafios e as perspectivas futuras para esse trabalho.

2 Revisão bibliográfica

O desenvolvimento desse trabalho engloba compreensão de práticas da engenharia de *software* e processamento digital de imagens. Dentre as práticas de engenharia de *software* utilizadas podem-se citar a qualidade de *software*, teste de *software*, em específico os testes de aceitação, regressão e exploratórios, além do processo de *software*. Compreensão sobre comparação de imagens é o ponto crucial sobre processamento digital de imagens que foi utilizado durante o projeto.

Este capítulo objetiva apresentar os conceitos presentes nesse trabalho. Tais conceitos são de suma importância para a compreensão, apresentação e o desenvolvimento do processo de testes utilizado. Portanto, se compõem de: Qualidade de Software, Testes de Software, Processo de Teste e Comparação de Imagens. Neste capítulo também estão presentes os trabalhos correlatos.

2.0.1 Qualidade de Software

A (ISO/IEC, 2001) é a norma que regulamenta os parâmetros para a avaliação da qualidade e como deve ser padronizada a avaliação de um produto de *software*. Segundo a norma, a qualidade de um produto de *software* pode ser percebida de várias formas e utilizando diversas abordagens, logo, seu intuito é estabelecer um modelo de qualidade que envolve diversos componentes. Esses componentes podem ser divididos em Qualidade do Processo e Qualidade do Produto. O processo de desenvolvimento de *software* é um conjunto de atividades realizadas por pessoas, com seus papéis, cujo objetivo é o desenvolvimento ou a evolução do *software* e os seus artefatos. Logo, práticas, artefatos e profissionais de qualidade, tendem a implicar em um sistema de *software* de boa qualidade. A qualidade do produto dá-se a relação entre atributos internos e externos, dados alguns atributos internos presentes no produto leva-se a aferir um atributo externo. Como, por exemplo, a manutenibilidade, um atributo externo de qualidade que é conseguido através dos atributos internos: profundidade da árvore de herança, complexidade ciclomática, tamanho do programa em linhas de código e extensão do manual de usuário.

Segundo (PRESSMAN; MAXIM, 2016) a qualidade de *software* representa a conformidade entre os requisitos funcionais e não funcionais, ou seja, os requisitos que foram especificados pelo cliente e os de características implícitas que não são declarados na documentação, mas são desejáveis em um *software*. Tais características são: desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenibilidade, interoperabilidade, testabilidade e portabilidade. Para poder mensurar quantitativamente tais características que são de certo modo subjetivas e relativas, são utilizadas métricas, onde posteriormente

pode ser realizado à análise e comparação desses, levando a possibilidade de quantificar e qualificar a qualidade do produto desenvolvido.

A qualidade de *software* é algo difícil de ser mensurado devido a sua natureza. Definir todas as especificações de um *software* é algo extremamente difícil, os requisitos funcionais descritos pelo cliente nem sempre são completos e claros. Há ainda os requisitos não funcionais, que apesar de não serem relatados pelo cliente são definitivos para a boa qualidade. Para que a qualidade possa ser mensurada e assim mostrada para os *stakeholders*, devem existir parâmetros onde há a comparação de valores, denominados métricas. As métricas são valores obtidos a partir de uma dada medição, definida de acordo com um parâmetro de qualidade no qual se deseja atingir. As métricas têm como principal objetivo servir como parâmetro de comparação entre os projetos, processos e documentação, conforme descreve (SOMMERVILLE, 2011).

2.0.2 Testes de Software

Teste de Software é um processo que faz parte do desenvolvimento de *software*, através dele busca-se melhorar a qualidade de um sistema, entregando o mesmo com poucas falhas e de modo que o usuário não as percebam. Desse modo, o objetivo dos testes é encontrar a maior quantidade de falhas e reportá-las à equipe de desenvolvimento, ainda em fase de concepção, para que as mesmas sejam corrigidas. O desenvolvimento de um produto de *software* com a ausência total de falhas é algo de imensa complexidade e impraticável, segundo (SOMMERVILLE, 2011). Ele ainda acrescenta, se um projeto buscasse entregar um *software* com a ausência de falhas, o projeto levaria vários anos fazendo com que seus custos não valessem o investimento. Mesmo assim, após décadas de projeto haveria uma grande possibilidade de ser descobertas algumas falhas com o decorrer do uso. A falha é a ocorrência de um erro, que por sua vez corresponde a um resultado não esperado ou fora das especificações. A ocorrência de um erro apenas é evidenciada se houver um defeito no código, algo não tratado pelo desenvolvedor, implementado de forma incorreta ou fora das especificações. SOMMERVILLE ainda descreve os testes como de natureza destrutiva, pois, seu principal objetivo é provocar um estado em que o *software* não funcione corretamente evidenciando um erro e identificando os defeitos presentes no sistema.

2.0.2.1 Testes de Aceitação

Teste de aceitação é do tipo teste de caixa preta, ou seja, é realizado sem análise do código da aplicação. Ele tem o propósito de avaliar a qualidade do produto de *software* e a sua usabilidade. É um tipo de teste com forte relação com o cliente, visto que ele reproduz a interação do usuário com o sistema. O teste de aceitação é realizado buscando mostrar se um sistema satisfaz ou não os critérios de aceitação propostos pelos requisitos funcionais

e não funcionais estabelecidos pelo cliente e pelo Gerente de Projeto. Ele deve focar em ações visíveis ao usuário, algo que será realizado por ele durante seu uso (PRESSMAN; MAXIM, 2016).

2.0.2.2 Automação de Testes

Segundo (MATHUR, 2013), testes manuais em sistemas grandes podem ser uma tarefa intensa e muitas vezes ineficaz. Executar centenas de vezes um cenário de teste, além de ser quase impraticável também é propenso a falhas humanas. A automação de testes é uma necessidade em sistemas grandes e oferece uma execução rápida e confiável de rotinas de testes de forma uniforme e idêntica. Isso faz com que *scripts* de testes sejam executados de maneira automatizada, tornando o processo de testes de regressão muito mais fácil, ágil e confiável.

Segundo (DUSTIN; RASHKA; PAUL, 1999) um processo de teste eficiente necessariamente deve ter incorporado a automação de testes. A automação de testes é um esforço de desenvolvimento que envolve estratégias, metas e definições de requisitos de testes, atividades de *design*, analise, desenvolvimento, verificação e validação. Com esse esforço se obtém agilidade, confiabilidade na execução dos cenários de testes e propicia ao analista de testes mais tempo para focar em outras abordagens de testes para assim encontrar uma maior ocorrência de falhas no produto de *software*.

2.0.2.3 Testes de Regressão

Segundo (DUSTIN; RASHKA; PAUL, 1999) é a prática de testar uma funcionalidade já validada após um novo incremento no sistema. Tendo como finalidade a verificação de conformidade do funcionamento de todas as funcionalidade já validadas de um sistema após uma nova versão do mesmo. Podendo essa nova versão conter apenas melhoria, correções de *bugs* ou conter uma novas funcionalidades, verificando a geração, ou não, de falhas após a alteração realizada em funcionalidades já validadas.

2.0.2.4 Testes Exploratórios

Segundo (BACH; BACH; BOLTON, 2006), testes exploratórios são uma categoria de testes não sistematizados, dessa forma desenvolver *scripts* para eles não é uma tarefa trivial. Essa prática de testes visa, através da experiência do analista de testes e um pouco de conhecimento sobre a aplicação, realizar testes não sistematizados, de modo a evidenciar falhas que os cenários de testes não cobrem e explorar o *software* atrás de mais falhas. É uma prática que deve ser utilizada para complementar os testes automatizados, pois, através da automação ganha-se tempo para explorar mais falhas.

2.0.3 Comparação de Imagens

Uma imagem pode ser representada por uma matriz de *pixels*, que por sua vez, são vetores de *bytes*. A comparação de imagens ou qualquer outra operação entre os elementos visuais de uma imagem, direta ou indiretamente estará lidando com os vetores de *bytes* que representam os *pixels*. A maneira mais simples de comparar se duas imagens são iguais, é por meio da comparação direta, ou seja, subtração de matriz de *pixels* entre essas imagens. Porém, essa abordagem é bastante limitada, pois, se a segunda imagem estiver dois ou três *pixels* para o lado o resultado será falso, algo que qualquer humano consideraria verdadeiro. Assim como qualquer variação de luminosidade e qualidade de uma imagem, pois, é algo que interfere em cada *pixel*. Por outro lado, é a abordagem mais simples e rápida de realizar comparações em imagens (BURGER; BURGE, 2016).

A comparação de imagem será aplicada na verificação de estados válidos da aplicação durante o processo de teste. A subtração de matriz de *pixels* foi a abordagem utilizada devido a uma eficiência computacional e pela simplicidade de implementação. Apenas dos limitantes essa abordagem satisfaz os requisitos propostos e atende completamente ao propósito.

2.0.4 Trabalhos Correlatos

Syed (SHAH et al., 2014) apresentam em sua pesquisa sistemática os pontos fortes e fracos sobre testes exploratórios e *scripts* de testes. Os pontos fortes e fracos foram apresentados sobre quatro categorias principais: Qualidade de teste; Natureza do processo; Custo benefício; e Satisfação do cliente. Pontos fortes apresentados sobre *scripts* de teste com base no trabalho realizado são, natureza do processo, qualidade de teste e satisfação do cliente, por serem repetíveis, reusáveis, garantir a qualidade de forma antecipada, apresentar melhor gerenciamento de riscos, não depender das habilidades dos analistas de testes e poder ser automatizado. Já os pontos fracos são: qualidade de teste por depender da qualidade, experiência e conhecimento do analista de testes para elaborar os casos de teste com uma boa qualidade e custo benefício pela questão da flexibilidade. Os pontos fortes dos testes exploratórios são custo benefício, natureza do processo e qualidade de teste, por apresentar pouco tempo destinado à documentação, melhor eficácia em detectar falhas, apresentar melhor regressão de testes e melhor detecção de falhas críticas. Já os pontos fracos são: qualidade de teste e satisfação do cliente, pois, a qualidade dos testes depende da experiência, do conhecimento e outras habilidades do analista de testes, e por não ser adequado para testes de aceitação, performance, incrementais, pois, reduz a responsabilidade e consequentemente a satisfação do cliente. Tal pesquisa sistemática serviu como base para implementação de uma abordagem híbrida de testes exploratórios executados por *scripts*, considerando a ISO/IEC 291191. Essa abordagem foi validada com uma pesquisa e de forma dinâmica na indústria.

(COLLINS et al., 2012) apresentam em seu trabalho o resultado de uma implementação de automação de testes usando ferramentas de testes de código aberto em dois projetos que utilizam a metodologia ágil SCRUM. Foram mapeadas as características dos projetos e elaborado um processo de teste e ferramenta para os projetos SCRUM. Não houve uma separação formal entre o time de desenvolvimento e o time de teste e foram usados casos de testes, histórias de usuário com base no *Product Backlog. JUnit, PHPUnit, Selenium, Jumeter, Fitnesse Test Tool* foram ferramentas utilizadas para codificação dos *scripts* e automação dos testes em abordagens como testes unitários, de funcionalidade, de sistema, de estresse, testes de aceitação, testes de carga e de integração. Em um dos projetos a cobertura dos testes de aceitação alcançou 87% e quando havia mudanças na interface alcançavam 50%. Já o outro projeto os testes unitários apresentaram cobertura de 70%, os de integração e os de requisitos não funcionais 100% e os de aceitação 40%. Os autores constataram que a automação de testes pôde funcionar de forma satisfatória em equipes que tem paciência e persistência em trabalhar de forma correta. Constataram também que a colaboração é um fator essencial para o sucesso, ferramentas de testes adequadas, estratégias de teste e metodologias ágeis também são. A automação de cada camada do *software*, quando possível e apenas em testes de aceitação provocou bons resultados. A automação em projetos ágeis deve ser simples, priorizando os testes regressivos, os testes de segurança e estresse na faze inicial do projeto reduz o risco e retrabalho. Deve-se usar também automação de testes para documentação e *feedbacks* informais.

Wellington (GONÇALVES et al., 2017), propõem em seu trabalho entender os fatores humanos: aspectos cognitivos, operacionais e organizacionais, que estão presentes no processo de teste e quais efeitos esses fatores podem causar na qualidade do processo. Para isso foi realizado uma pesquisa com profissionais da área de teste como, analista, arquiteto, gerente, líder e executor dos testes e outros. O estresse e conflitos com desenvolvedores foram os dois fatores que apresentaram maior impacto, com alguns profissionais da área ainda acrescentaram que é um trabalho bastante exaustivo, diminuindo sua autoestima, dando a sensação monótona e afetando a forma como eles trabalham. Através dos dados coletados pelo estudo os autores concluíram que os fatores humanos influenciam no processo de teste devido sua natureza, pouco reconhecimento dos colaboradores e isso provoca impactos negativos no processo de teste e consequentemente na qualidade do *software*.

Cuixiong Hu e Iulian Neamtiu (HU; NEAMTIU, 2011) propuseram um modelo de automação de testes de interface em aplicativos móveis da plataforma Android. Primeiro realizaram um estudo empírico sobre as naturezas dos *bugs* de interfaces e posteriormente elaboraram uma técnica de detecção pela geração de casos de testes automatizados. Eles combinaram a geração de casos de testes automatizados com ferramentas de geração de eventos pela análise de arquivo de *log*. Esse arquivo era coletado durante a execução dos casos de testes do arquivo de *log* do sistema e foi útil para prover informações mais detalhadas da aplicação. A geração de eventos automáticos foi considerada uma poderosa

técnica para verificação de interfaces de uma aplicação. Eles usaram dez aplicativos Android populares da *Play Store*. Através dessa abordagem eles conseguiram replicar todos os *bugs* relatados e ainda reportar *bugs* ainda não identificados.

3 Desenvolvimento

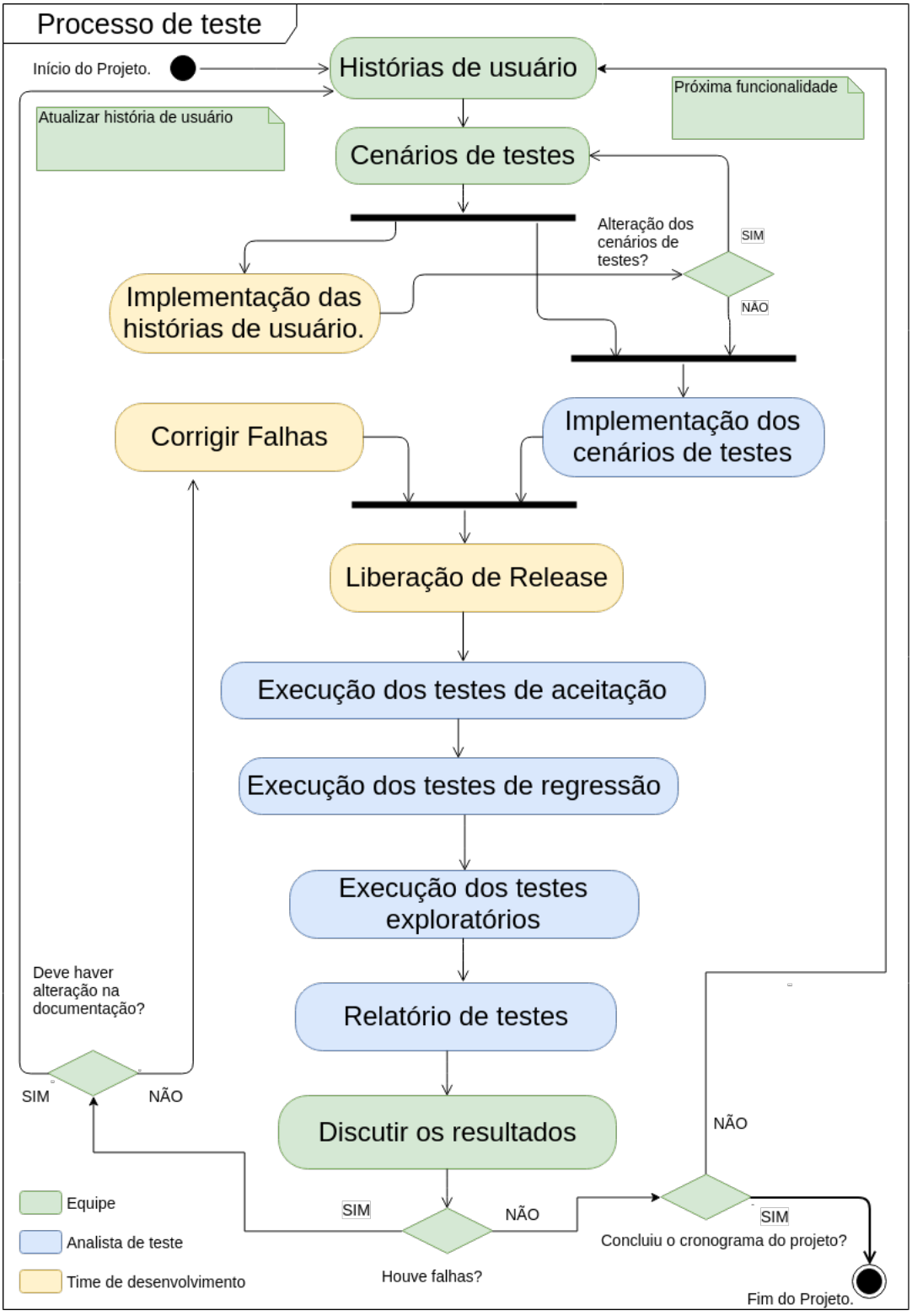
Este capítulo descreve como era o fluxo de desenvolvimento e execução dos testes utilizados pelo projeto MobMine, envolvendo documentações e ferramentas antes da utilização da comparação de imagens para validação dos cenários de testes. Serão descritos os diferentes momentos da implantação do processo. Por fim, também será descrito de maneira detalhada a implementação da comparação de imagens.

3.1 O momento anterior

O desenvolvimento dos testes no projeto MobMine até a utilização de comparação de imagens seguia o processo de teste vigente no laboratório iMobilis, ilustrado na Figura 1. Ele é composto por documentações que serviram de guia para implementação, tanto dos testes quanto das funcionalidades, relatórios de testes, uma ferramenta para codificação e automação de testes e o ciclo de testes.

O ciclo do processo de testes ocorria da seguinte maneira para as funcionalidades que não utilizavam a ferramenta de desenho 2D: Para cada nova funcionalidade o time de desenvolvimento e o analista de testes elaboravam juntos a documentação de histórias de usuário e posteriormente os cenários de testes de acordo com seus requisitos da funcionalidade. Posteriormente era implementada a funcionalidade pelo time de desenvolvimento e os cenários de testes pelo analista de teste. Com a funcionalidade já implementada o time comunicava o analista para que pudessem ser realizados os testes. Primeiro eram realizados os testes da funcionalidade implementada, posteriormente os testes regressivos e por fim uma bateria de testes exploratórios. Após esse processo o analista de testes elaborava um relatório apresentando os resultados dos testes. Esse relatório continha informações da versão liberada, como o nome da versão, as funcionalidades e falhas corrigidas, além do resultado dos testes. Ele sempre era apresentado e discutido nas reuniões semanais que ocorriam no iMobilis. Quando eram detectadas falhas, o analista comunicava ao time para que fosse realizada a correção das mesmas. Em uma versão futura o time de desenvolvimento comunicava ao analista de testes que foram corrigidas as falhas reportadas e ele realizava a verificação por meio dos testes.

Figura 1 – Diagrama do processo de teste utilizado no iMobilis



Fonte: Elaborado pelo autor.

Para partes da aplicação que utilizam a ferramenta de desenho 2D, eram realizados os testes de aceitação de maneira manual, além de testes exploratórios. Algo que agregava valor ao produto, porém, era propenso a falhas por ser uma grande quantidade de cenários de testes e sendo crescente essa quantidade com a evolução do projeto. Algo que com o tempo chegou a ser, gasto, semanalmente sete horas, tempo considerado alto, para execução tanto dos cenários de testes manuais quando os exploratórios.

3.1.1 Documentação

A documentação utilizada para guiar o desenvolvimento dos testes e da aplicação foram as histórias de usuários e cenários de testes. Esses artefatos já eram utilizados no laboratório como documentação de requisito dos projetos. As histórias de usuário descrevem a forma como o usuário utilizará o sistema, contendo uma estrutura direcionada simples, que contém: um usuário, com um papel, querendo realizar uma ação, que corresponde a uma funcionalidade, para alcançar um objetivo. Como forma de padronização, foi utilizado o seguinte formato para descrição das histórias de usuário:

COMO [*papel*];
EU GOSTARIA [*funcionalidade*];
PARA CONSEGUIR [*objetivo*];

Os cenários de testes são escritos com base nas histórias de usuário, onde cada história deve ter no mínimo dois cenários de testes, um cenário principal e outro alternativo. O cenário principal apresentando um caso de sucesso, onde tudo ocorre como esperado, entradas digitadas de forma correta e nada interferindo de forma negativa no sistema. Já o cenário alternativo apresenta o acontecimento de algo que impede a conclusão da história, uma falha na conexão, dados inseridos errados ou algo semelhante. São escritos para casos possíveis de ocorrer e que o software deve saber se comportar de maneira correta. Podem haver vários cenários alternativos, isso dependerá de como está projetado a aplicação. A estrutura dos cenários de testes apresenta uma pré-condição, que é uma condição para a realização de uma funcionalidade, como, por exemplo, a necessidade de conexão com a internet. Além da pré-condição deve haver um evento, que é o passo que o usuário realizará para alcançar seu objetivo, que é o resultado esperado e que também deve ser evidenciado.

DADO [*pré-condição*];
QUANDO [*evento*];
ENTÃO [*resultado esperado*];

E [*extensão*];

MAS [*extensão*];

Um relatório de testes foi utilizado de modo que o analista pudesse reportar ao time de desenvolvimento as falhas e os possíveis erros que as ocasionavam. Nesse relatório era apresentado o *changelog* da *release*, os resultados dos testes dessa *release* e até mesmo sugestões sobre melhorias de interface e usabilidade. Essas sugestões eram debatidas pela equipe, bem como os resultados dos testes.

Foi criado por toda a equipe um padrão de codificação para que os desenvolvedores seguissem. Esse artefato padroniza a forma na qual são definidos os nomes de botões, de *views*, de identificadores, dentre outros elementos da aplicação. Além de padronizar a codificação dos aplicativos móveis, ele auxilia o desenvolvimento dos cenários de testes quando há a necessidade da referência de um elemento gráfico e na manutenção e evolução do sistema. Além de tornando mais fácil a localização de um elemento no código por meio de mecanismos de busca presente na IDE.

3.1.2 Ferramenta

A ferramenta utilizada para implementar e executar os testes automatizados foi o Robotium, que é um *framework* desenvolvido a partir do JUnit. Por meio dele é possível a manipulação de elementos de interface de maneira simples, ele também provê funções que auxiliam no processo de teste, fazendo com que a implementação dos cenários de testes seja o mais alto nível possível, trazendo agilidade na implementação dos testes. O Robotium é facilmente incorporado ao Android Studio, IDE utilizada pelos engenheiros de software do laboratório para o desenvolvimento de aplicativos Android, sendo incorporado ao projeto simplesmente adicionando os comandos nos seguintes blocos de código no Gradle[1]:

```
1
2       defaultConfig {
3           testInstrumentationRunner
4               "android.support.test.runner.AndroidJUnitRunner"
5       }
6
7       dependencies {
8           testCompile 'junit:junit:4.12'
9
10          androidTestCompile
11              'com.jayway.android.robotium:robotium-solo:5.6.3'
12
```

[1] <https://www.androidpro.com.br/gradle/>

```
        androidTestCompile
            'com.android.support.test:runner:1.0.1'
        androidTestCompile
            'com.android.support.test:rules:1.0.1'
    }
```

O Robotium possui uma linguagem simples, como outras ferramentas de mesmo propósito e comparada ao JUnit, deixando a implementação dos testes muito rápida e intuitiva. Como pode ser visto no trecho de código abaixo, que é um demostrativo da ferramenta disponível no cite <http://www.vogella.com/tutorials/Robotium/article.html#robotium_usage>. A execução dos testes pode ser feita método por método, uma classe inteira ou até mesmo pacotes inteiros. Forma que facilita a execução e proporciona maior agilidade para executar suítes de testes regressivos.

```

package de.vogella.android.test.target.test;

import junit.framework.Assert;
import android.test.ActivityInstrumentationTestCase2;

import com.robotium.solo.Solo;

import de.vogella.android.test.target.SimpleActivity;
import de.vogella.android.test.target.SimpleListActivity;

public class SimpleActivityTest extends
        ActivityInstrumentationTestCase2<SimpleActivity> {

    private Solo solo;

    public SimpleActivityTest() {
        super(SimpleActivity.class);
    }

    public void setUp() throws Exception {
        solo = new Solo(getInstrumentation(), getActivity());
    }


    @Override
    public void tearDown() throws Exception {
        solo.finishOpenedActivities();
```

```
29      }
30  }
31
32  // check that we have the right activity
33  solo.assertCurrentActivity("wrong activity", SimpleActivity.class
        );
34
35  // Click a button which will start a new Activity
36  // Here we use the ID of the string to find the right button
37  solo.clickOnButton(solo.getString(R.string.button1));
38  // Validate that the Activity is the correct one
39  solo.assertCurrentActivity("wrong activity", SimpleListActivity.
        class);
40  solo.clickInList(1);
41  // searchForText has a timeout of 5 seconds
42  assertTrue(solo.waitForText("Android")); // Assertion
43  solo.clickInList(2);
44  assertTrue(solo.waitForText("iPhone")); // Assertion
45  solo.clickInList(3);
46  assertTrue(solo.waitForText("Blackberry")); // Assertion
47  solo.goBack();
48  solo.clickOnButton("Button2");
49  solo.clickOnButton("Button3");
50
51  // open the menu
52  solo.sendKey(Solo.MENU);
53  solo.clickOnText("Preferences");
54  solo.clickOnText("User");
55  solo.clearEditText(0);
56  Assert.assertTrue(solo.searchText(""));
57  solo.enterText(0, "http//:www.vogella.com");
58  Assert.assertTrue(solo.searchText("http//:www.vogella.com"));
59  solo.goBack();
```

3.1.3 Métricas

Com o intuito de acompanhar, comparar e medir o impacto da implantação e evolução do processo de testes e da comparação de imagens dentro do processo de testes, foram utilizadas métricas que fossem possíveis de mensurar, através de dados coletados, como está sendo a evolução e qual o impacto provocado pelo processo de teste no projeto. Com base na revisão bibliográfica sobre qualidade de software apresentada no capítulo

anterior, foram selecionadas as métricas nas quais seriam coletados os dados e comparados ao longo do projeto e posteriormente poderia ser comparado também a outros projetos. As métricas foram escolhidas utilizando alguns classificadores da lista de planejamento proposto por (HARTMANN; DYMOND, 2006). As métricas escolhidas foram:

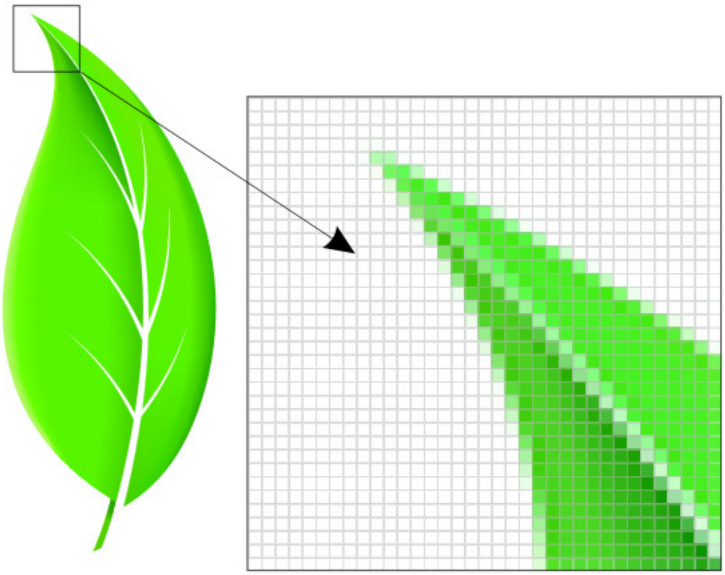
- **Quantidade de cenário de teste:** Tem como objetivo identificar e mensurar o tamanho do aplicativo, quanto maior o número de cenários de teste, maior o número de funcionalidades e maior é o aplicativo. É apenas o total de cenários de testes.
- **Quantidade de cenários de teste por funcionalidade:** Tem como objetivo acompanhar a quantidade de testes produzidos. É medida através da razão entre o número total de cenários de teste pelo número total de funcionalidades.
- **Tempo de desenvolvimento dos códigos de teste de aceitação:** Tem como objetivo identificar a complexidade da codificação dos testes. É medida pelo tempo de desenvolvimento dos códigos de teste.
- **Tempo de execução do conjunto do teste de aceitação:** Objetiva medir o tempo de execução do conjunto de testes. O próprio Android Studio fornece essa métrica como parâmetro de medição.
- **Fator de teste de aceitação:** Tem como objetivo verificar o esforço para criação dos códigos de teste e verificar sua evolução. Ela é medida através da divisão do total de linhas de código de teste pelo total de linhas de código do aplicativo.
- **Quantidade de ocorrência:** Tem como objetivo verificar quantos cenários de teste falharam e estão esperando correção para serem retestados. É medida pela quantidade de cenários que esperam ser retestados.
- **Quantidade de Falhas:** Tem como objetivo verificar a quantidade de falhas que não puderam ser corrigidas. É medida pela quantidade de funcionalidades que não passaram no teste e não puderam ser corrigidas.
- **Porcentagem de assertivas passando e falhando:** Tem como objetivo verificar a porcentagem de assertivas de teste passando ou falhando. É medida através da porcentagem da divisão do número de assertivas passando/falhando pelo total de assertivas. Entende-se por assertivas o que deve ser testado e não se tem certeza se passou ou não. Esse jargão é usado devido ao fato que mesmo uma aplicação passando ou não no teste em um certo momento, posteriormente com a introdução de uma nova funcionalidade o resultado pode ser diferente.
- **Quantidade de funcionalidades testadas e entregues (Running Tested Features ou RTF):** Tem como objetivo saber a quantidade de funcionalidades prontas.

É medida pela subtração do total de funcionalidades pelo número de funcionalidades que passou no teste.

3.2 Comparação de imagens

A técnica utilizada para a realização da comparação de imagens foi escolhida primeiramente por ser a abordagem mais simples e por ser bastante confiável, caso sejam seguidos alguns critérios. A subtração de matriz de pixel é realizada através da representação de duas imagens no formato de matriz, como pode ser observado simbolicamente na Figura 3, ou vetores de pixel. Subtração de vetores consiste em pegar o valor de cada posição do vetor A e subtrair pelo valor da mesma posição do vetor B, se o resultado for zero, significa que esses valores são iguais, caso contrário são diferentes. Para duas imagens serem consideradas iguais elas devem ter a mesma quantidade de *pixels* e a subtração de cada *pixels* deve resultar uma imagem em branco, ou seja, apresentar todos os *pixels* com valores zeros. No entanto, ao se comparar duas ou mais imagens através desta técnica, elas precisam ter a mesma quantidade de *pixels*, caso contrário a subtração acarretará um erro do programa. Segundo (BURGER; BURGE, 2016) essa é a abordagem mais simples e apresenta o melhor desempenho computacional em relação as outras abordagens dessa natureza.

Figura 2 – Representação de uma imagem em um matriz

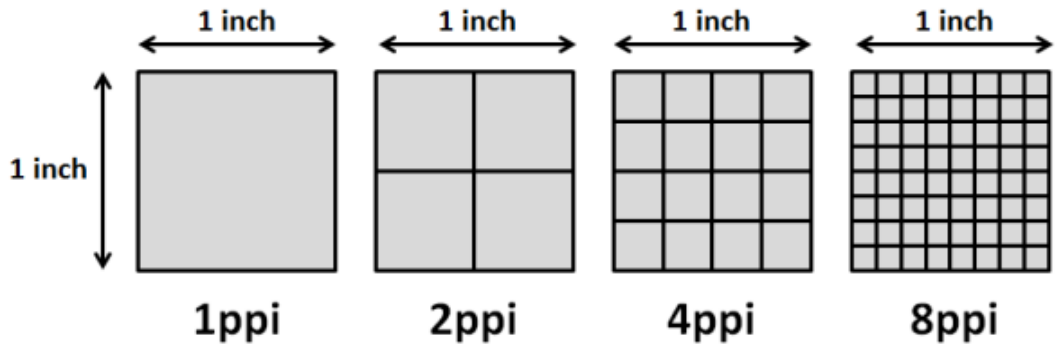


Fonte: <https://www.coreldraw.com/static/cdgs/images/content/products/cdgsx5/tutorials/tut04/CDGSX5_Tut04_WebGraphics.jpg>

No projeto MobMine são utilizados dois modelos de dispositivos *tablets*, ambos da

mesma marca e com as mesmas configurações, porém, um tem tela de sete polegadas e o outro tem a tela de dez polegadas. Mesmo a tela dos dispositivos apresentando tamanhos diferentes ambas contêm a mesma quantidade de *pixels*, no entanto, a densidade de *pixel* do *tablet* menor é maior, fazendo com que ele apresente uma imagem ligeiramente melhor, diferença que é quase imperceptível a olhos humanos. A densidade de *pixels* corresponde a quantidade de *pixels* que uma determinada área da tela de um dispositivo contém. Como pode ser visualizado melhor na Figura 3, quanto maior a densidade melhor será a qualidade da imagem, caso estejamos comparando dispositivos de mesma tecnologia.

Figura 3 – Densidade de pixel



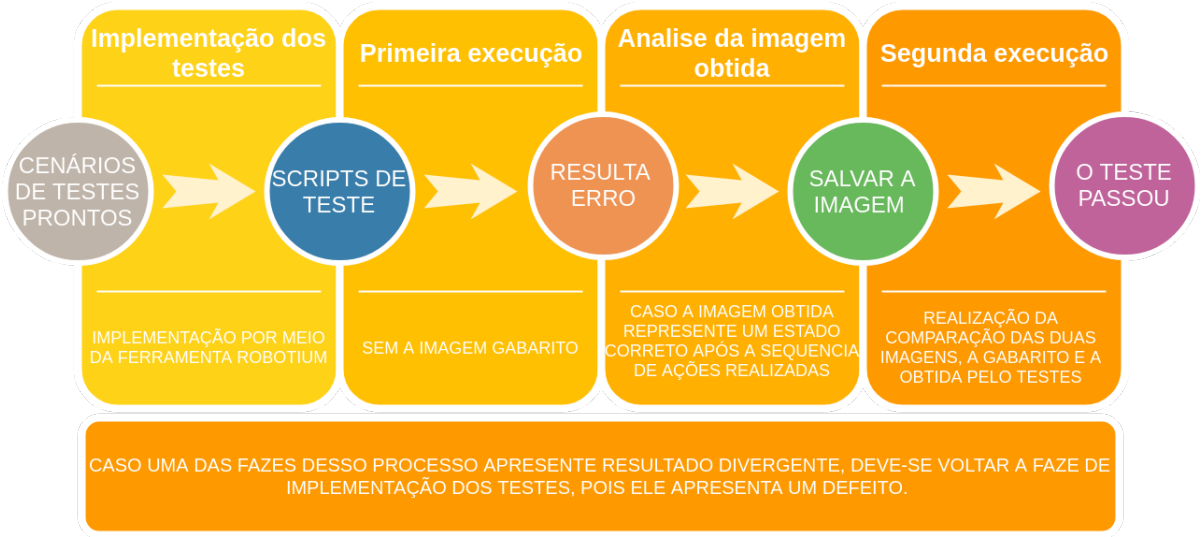
Fonte: <https:/https://img3.ibxk.com.br/2014/08/18/18182134551531.png?w=700>

3.3 Comparação de imagens dentro do processo de teste

O processo de comparação de imagens proposto ocorre da seguinte maneira, como pode ser visto na Figura 4. Após os cenários de testes definidos, são gerados os *scripts* de teste no Robotium, contendo o passo a passo da execução de uma funcionalidade, no final de cada cenário de teste é realizado uma captura de tela através do *framework* Robotium. Esse processo é feito de maneira supervisionada verificando a legitimidade do cenário de testes e a ação realizada pelo *framework* de automação. Também é analisada a legitimidade da representatividade do estado correto apresentado na imagem.

O Robotium contém um método que realiza a captura de tela e armazena a imagem gerada em uma pasta chamada *Robotium-screenshot* no formato *.png* na raiz do dispositivo. Os parâmetros de entrada dessa função podem ser o nome no qual se deseja salvar o arquivo gerado, há opções também, de passar como parâmetro a resolução da captura de tela dentre outros parâmetros. Para o contexto e proposito deste projeto, foi utilizada apenas a função na qual passamos o nome da imagem a ser gerado e salvo.

Figura 4 – Fazes do processo de testes utilizando a comparação de imagens



Fonte: Elaborado pelo autor.

Após a captura da imagem, pelo algoritmo de teste, a mesma deve ser enviada juntamente com a imagem que representa o estado esperado, após a mesma sequência de ações, para o algoritmo que realizará a comparação. A comparação de imagens dentro do desenvolvimento Android, foi realizada através do objeto nativo Android BITMAP[2]. As imagens **previstas** e as **obtidas** eram transformadas em vetores de pixel, onde posteriormente seriam subtraídos. A função que realiza a comparação de imagens retorna *true* se as imagens são iguais e *false* se há alguma divergência entre elas.

Durante a implementação de um cenário de teste, o analista de testes, executará os testes de modo a gerar a primeira imagem. Como ainda não se tem a imagem gabarito, o algoritmo não conhece a imagem esperada e o primeiro teste desse cenário acarretará em um erro. Porém, após essa primeira execução uma imagem será gerada e ela é analisada e verificada manualmente pelo analista de teste. Caso essa imagem represente um estado correto da aplicação após a sequência de ações realizada, ela é considerada correta e é movida para uma pasta denominada gabarito, que conterá todas as imagens que representam um estado correto. Com a imagem gabarito do cenário implementado já salva na pasta, ao executar novamente o cenário de teste, será comparada a imagem da pasta gabarito com a imagem gerada pelo teste. Caso ambas sejam idênticas o teste apresenta sucesso para o cenário de teste executado. Na presença de erro em uma das fazes desse processo, se observa que há uma falha nos testes e evidentemente um defeito no *script* de teste, indicando que os testes devem ser corrigidos.

As imagens são identificadas, pelo algoritmo comparador de imagem, pelo nome.

[2] <https://developer.android.com/reference/android/graphics/Bitmap.html>

Toda imagem contém o nome do cenário de testes na qual ela pertence. A imagem esperada estará na pasta gabarito na raiz do dispositivo e a imagem a ser comparada estará na pasta *Robotium-screenshot*. Desta forma, as imagens foram sendo geradas para cada cenário de testes.

Outra abordagem utilizada foi a automação de testes exploratórios, que segundo (SHAH et al., 2014), é uma abordagem híbrida e prove maior ganho de tempo e confiabilidade. Foram realizados esses testes híbridos para verificar os estados em diferentes arquivos que eram usados de *inputs* de dados específicos da regra de negócio da aplicação. Cada arquivo teve que ser analisado e posteriormente, sempre que uma nova *release* era disponibilizada esses testes híbridos eram realizados de maneira supervisionada, onde o analista de testes acompanhava visualmente a execução dos testes e reportava as ocorrências de possíveis falhas.

Com proposito de documentar o processo utilizado e manter o gabarito em um local que toda equipe possa ter acesso. O gabarito, assim como os cenários de testes foram arquivados na *wiki* do laboratório, em uma página privada para os membros do projeto. Com exceção dos testes exploratórios que não apresentam documentação.

4 Resultados

A Tabela 1 apresenta o estado do projeto MobMine nos Sprints 10, 14 e 15. Foram utilizados esses Sprints por representarem grandes marcos do projeto. O Sprint 10 foi quando a primeira *release* funcional foi entregue ao *Product Owner* (PO). O Sprint 14 representa uma segunda grande entrega, apresentando funcionalidades muito complexas e relevantes ao projeto e o Sprint 15 por ser a concretização do processo de comparação de imagens dentro do processo de testes e ser o último a ser concluído. Tendo em vista que o processo de testes se deu início no Sprint 6 e os dados dos Sprints não mencionados não apresentarem diferenças significativas, serão apresentados, discutidos e comparados somente os Sprints anteriormente citados.

Como pode ser observado na Tabela 1 sobre o Sprint 15 em relação aos demais, o fator de teste de aceitação aumentou para aproximadamente 23%. Valor que possibilitou a detecção de algumas falhas ainda no processo de desenvolvimento da aplicação, indicando um aumento considerável da cobertura dos testes. A quantidade de cenários de testes por funcionalidade está acima do mínimo. A quantidade de cenários de teste aumentou-se ligeiramente devido à continuidade do projeto e o tempo gasto para a execução dos testes teve um aumento superior a 100%. Esse aumento refletiu a automação dos testes, que anteriormente apenas 40% eram automatizados e após a utilização da comparação de imagens, foi possível alcançar os 100%.

Tabela 1 – Comparação entre os Sprints do projeto em relação às métricas propostas

Categorias	Métricas	Sprint 10	Sprint 14	Sprint 15
Aceitação	Quantidade de cenários de testes	104	110	130
Aceitação	Quantidade de Cenários de teste por funcionalidade	2,17	2,15	2,78
Aceitação	Tempo de desenvolvimento dos scripts de teste de aceitação	40h	80h	80h
Aceitação	Tempo de execução do conjunto de teste de aceitação	15’	45’	2:30h
Aceitação	Fator de teste de aceitação	13,5%	15,56%	22,87%
Geral	Quantidade de ocorrência	0	18	19
Geral	Quantidade de Falhas	0	2	6
Geral	Porcentagem de assertivas passando	100%	81,82%	91%
Geral	Porcentagem de assertivas falhando	0%	18,18%	9%
Geral	Quantidade de funcionalidades testadas e entregues	48/48	51/51	60/60

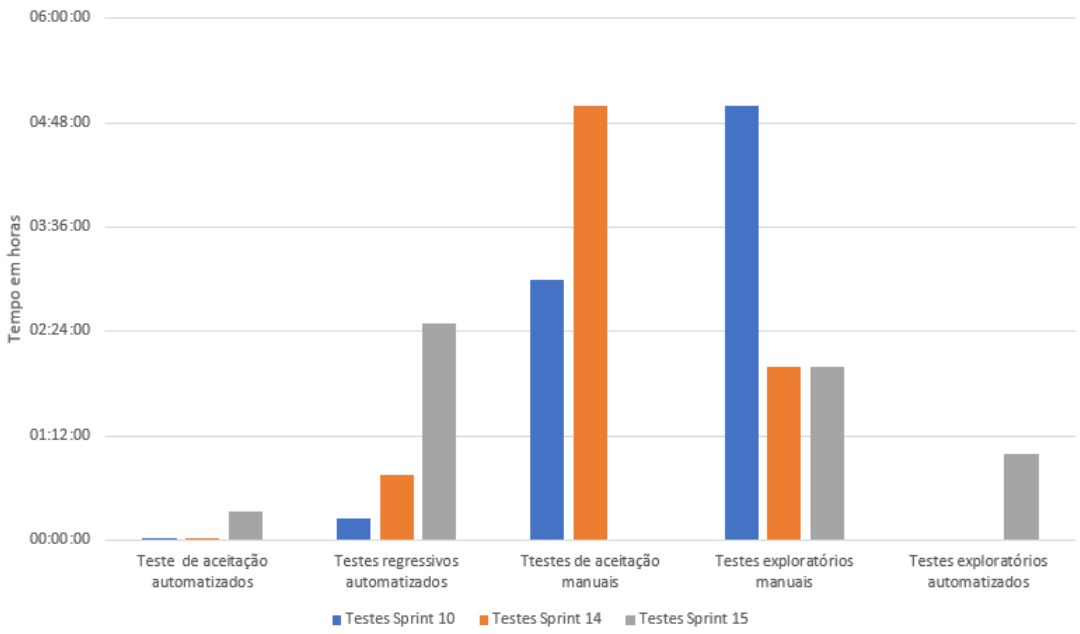
Fonte – Produzido pelo autor.

Os testes automatizados antes da comparação de imagens gastavam aproximadamente 15 minutos no Sprint 10 e 45 minutos no Sprint 14, como pode ser visualizado na

Tabela 1. Agora, com a automação completa dos cenários de testes gastam em média duas horas e meia. Entretanto, como pode ser analisado na Figura 5, o tempo gasto para a execução dos testes de aceitação manuais, que vinham crescendo, após a utilização do processo de comparação de imagem chegou a zero no Sprint 15. A execução dos testes regressivos automatizados teve um aumento de tempo de aproximadamente quatro vezes, algo positivo pois isso se deve ao fato de que todos os cenários de testes planejados foram automatizados. Os testes exploratórios manuais se mantiveram iguais comparados ao Sprint anterior.

Atualmente com a automação dos testes regressivos, o analista de testes aguarda o término de sua execução, podendo realizar outras atividades em paralelo, e ao fim da execução apenas verifica os resultados. Caso não haja falhas, ele prossegue para a execução dos demais testes. Mas caso haja, apenas os cenários de teste referentes a essas falhas são executados novamente de forma isolada, para que possa ser identificado o motivo da falha e assim comunicá-la a equipe de desenvolvimento.

Figura 5 – Gráfico comparativo entre o tempo gasto na execução de cada tipo de testes



Fonte: Produzido pelos autores.

Algumas funcionalidades que envolvem a ferramenta de desenho 2D são bastantes sensíveis, pequenas falhas como exclusão de um ponto do desenho comprometem toda a aplicação. Prova disso foram três funcionalidades que apresentaram grandes mudanças nos desenhos gerados e cinco *bugs* que levaram toda a aplicação a estados errôneos. Apesar dessas funcionalidades e *bugs* não terem afetado as outras funcionalidades como foi observado nos testes regressivos, elas levam o programa a estados inválidos, algo que

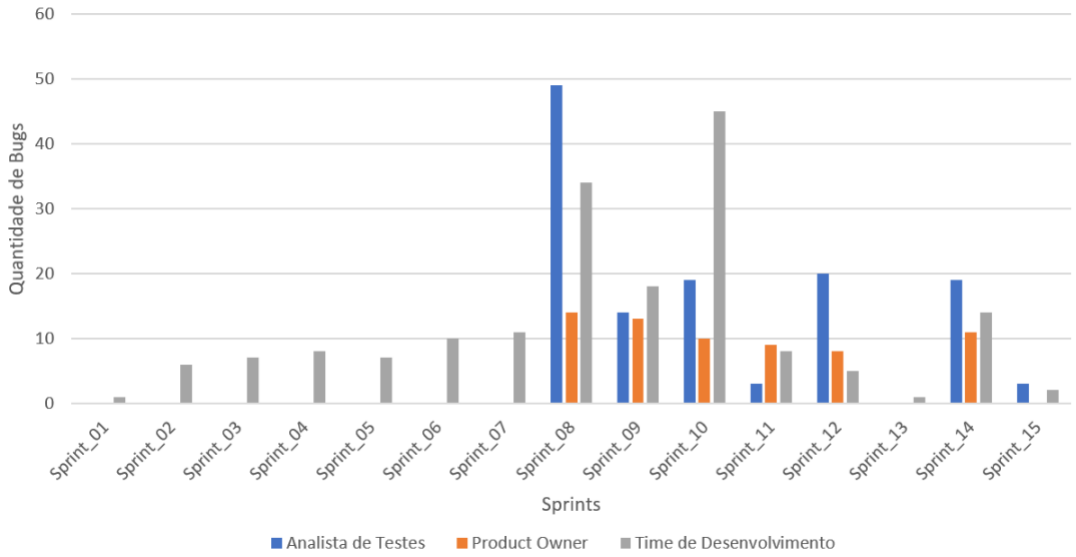
compromete o sistema por completo. Isso fez com que o processo de testes evoluísse de maneira lenta. A verificação das imagens esperadas tinha que ser feita com muita atenção, fazendo com que o mesmo cenário de teste fosse repetido inúmeras vezes e a verificação das imagens decorrente de uma funcionalidade demorassem em média uma hora. Quatro vezes o tempo gasto antes, tempo esse não esperado pela equipe do projeto. Uma abordagem que é aparentemente simples de ser desenvolvida acabou requerendo um cuidado muito grande. Por tais funcionalidades serem consideradas ponto chave da regra de negócio da aplicação e requerer uma alta precisão. Entretanto, mesmo com a necessidade dessa atenção maior no momento de implementação dos testes, através da automação desse processo, foi ganho um tempo na execução dos testes de regressão. Além de agregar uma maior qualidade ao produto decorrente do projeto.

A abordagem de realizar testes híbridos fez com que ações rotineiras feitas durante testes exploratórios fossem executadas de forma automatizada, ou seja, de forma rápida, sistematizada e independente. Através disso foi possível identificar diversos *bugs* sobre instâncias de dados específicos da aplicação. Os testes manuais exploratórios passaram a gastar duas horas e os exploratórios manuais uma, um aumento de 50% do tempo gasto para essa categoria de testes, e trouxe uma maior confiabilidade por parte deles serem automatizados.

A Figura 6 apresenta a disposição do total de falhas reportadas em cada Sprint pelo analista de testes, pelo time de desenvolvimento e pelo *Product Owner* (PO). O processo de teste iniciou-se no Sprint 6, porém nos dois primeiros Sprints o analista de testes, dedicou seu tempo a estudar o projeto, conhecer a aplicação e o seu propósito. Já no Sprint 8 foram reportadas pelo analista de testes as primeiras falhas, e coincidentemente pelo *Product Owner* (PO) que teve seu primeiro contato com uma versão beta da aplicação. Essa versão beta teve o propósito de coletar algumas falhas e também melhorias. No Sprint 10 o time de desenvolvimento se esforçou ao máximo para concluir o cronograma e entregar a primeira versão estável para o *Product Owner* (PO), logo como pode ser observado na Figura 6, eles encontraram a maior ocorrência de falhas no Sprint 10. Esse esforço do time de desenvolvimento refletiu no Sprint 11, onde foram reportadas poucas falhas. Por meio da Figura 6, também pode-se observar que o processo de testes utilizando a abordagem híbrida, que foi iniciada no final do Sprint 11, levou a um aumento de cinco vezes o número de falhas encontrado no Sprint 12. Onde teve o início de introduções de algumas funcionalidades que afetaram diversas partes da aplicação, esse aumento também se deve a parte dos testes regressivos que identificaram algumas propagações de funcionalidades incorporadas ao projeto.

Por fim, após a utilização da comparação de imagens, o tempo gasto no processo de testes resultou em poucas falhas encontradas. Porém, levando em conta que o Sprint 15 é o Sprint posterior a uma entrega, comparando ele ao Sprint 11 que também foi um Sprint

Figura 6 – Falhas reportadas por Sprint, e quem as reportou



Fonte: Elaborado pelo autor.

posterior a um entrega, observamos que já era esperado a ocorrência de poucas falhas. O Sprint 15 foi escolhido para início da inserção da comparação de imagens dentro do processo de testes por ser um Sprint mais tranquilo, após a entrega de uma funcionalidade, e as atividades do time de desenvolvimento foram apenas correções de falhas específicas reportadas ainda no Sprint 14 pelo *Product Owner* (PO).

5 Conclusão

O processo de comparação de imagens dentro do processo de testes conseguiu ser concretizado, proporcionando a execução automatizada de testes regressivos nas funcionalidades que envolvem a ferramenta de desenho 2D. O analista de teste teve um ganho de tempo de 100% na execução dos testes de regressão e uma melhor distribuição na execução dos testes, após a o método proposto. O processo de execução de todos os testes gasta atualmente seis horas por *release*, onde apenas duas horas e meia necessitam de uma atenção do analista de testes, um ganho de tempo de mais de 200% para essa finalidade. A utilização dos testes híbridos além de trazer uma maior confiabilidade no processo de execução dos testes exploratórios, trouxe um aumento na quantidade de testes exploratórios executados no mesmo tempo destinado a execução desses. Após o gabarito das imagens de testes ter sido completado e os cenários de testes terem sido implementados, o processo de testes de regressão pode ser executado de forma automatizada e independente. Fazendo com que sua execução fosse paralela a outras atividades do processo de teste, e o analista de testes apenas observasse os resultados e trabalhasse em cima deles. Através dessas técnicas o analista de teste teve um ganho enorme de tempo no processo de execução dos testes, esse tempo ganho será destinado ao planejamento de cenários de testes mais elaborados e ao planejamento e execução de testes exploratórios. Sempre visando encontrar a maior quantidade de falhas ainda na fase de desenvolvimento.

Até o presente momento com os quinze Sprints completados e com os resultados obtidos e apresentados no capítulo anterior, pode-se constatar que o processo de teste automatizado além de ter ocorrido com sucesso, foi um ponto muito importante do processo de testes. Após o processo de automação dos testes de aceitação e regressão por completo e utilização dos testes híbridos, o conjunto de testes passaram a ser executados mais rápidos sobrando mais tempo para o analista de teste poder realizar testes manuais exploratórios. Além de aumentar a cobertura dos testes, desse modo pode encontrar mais falhas. Através dos testes exploratórios foi possível provocar diversas ocorrências de falhas, que puderam ser passadas ao time de desenvolvimento como erros da aplicação, ou informalmente falando *bugs*.

Por fim a implantação do processo de testes no projeto ocorreu e ainda ocorre de maneira incremental, suprindo as necessidades e acompanhando a evolução do projeto de maneira satisfatória. A equipe do projeto e o *Product Owner* (PO) encontram-se satisfeitos com os resultados obtidos, por ter a primeira e a segunda *release* entregue no prazo, contemplando todos os requisitos, e esses funcionando como esperado e superando as expectativas do *Product Owner* (PO). Em virtude dos resultados obtidos pode-se afirmar que o produto de software oriundo do projeto vem sendo desenvolvido com um processo

que vem suprindo as lacunas do desenvolvimento e encaminhando-o para se tornar uma aplicação de boa qualidade. Sendo o processo de comparação de imagens uma técnica que trouxe bastante prestígio, confiabilidade e ganho de tempo para o processo. Aumentando ainda a qualidade do produto de software gerado pelo projeto.

Devido a ferramenta de desenho 2D ser sensível a alterações, houve a dificuldade de implementação dos cenários de testes. Com o acréscimo de algumas funcionalidades os cenários de testes decorrentes das funcionalidades que utilizam a ferramenta, tiveram que ser refeitos três vezes ao longo do processo de implementação. Algo que trouxe um retrabalho enorme para o analista de testes, fazendo com que esse gastasse semanas refazendo esses cenários de testes. Como trabalho futuro pretende-se utilizar a comparação de imagens juntamente ao processo de testes híbridos. Algo que diminuirá ainda mais o tempo gasto pelo analista de testes para execução desses.

Referências

BACH, J.; BACH, J.; BOLTON, M. *Exploratory Testing Dynamics*. [S.l.]: Version, 2006. Citado na página 19.

BURGER, W.; BURGE, M. J. *Digital image processing: an algorithmic introduction using Java*. [S.l.]: Springer, 2016. Citado 2 vezes nas páginas 20 e 30.

COLLINS, E. F. et al. Software test automation practices in agile development environment: An industry experience report. In: IEEE PRESS. *Proceedings of the 7th International Workshop on Automation of Software Test*. [S.l.], 2012. p. 57–63. Citado na página 21.

CRISPIN, L.; GREGORY, J. *Agile testing: A practical guide for testers and agile teams*. [S.l.]: Pearson Education, 2009. Citado na página 14.

DUSTIN, E.; RASHKA, J.; PAUL, J. *Automated software testing: introduction, management, and performance*. [S.l.]: Addison-Wesley Professional, 1999. Citado 2 vezes nas páginas 14 e 19.

GONÇALVES, W. F. et al. The influence of human factors on the software testing process: The impact of these factors on the software testing process. In: IEEE. *Information Systems and Technologies (CISTI), 2017 12th Iberian Conference on*. [S.l.], 2017. p. 1–6. Citado na página 21.

HARTMANN, D.; DYMOND, R. Appropriate agile measurement: using metrics and diagnostics to deliver business value. In: IEEE. *Agile Conference, 2006*. [S.l.], 2006. p. 6–pp. Citado na página 29.

HU, C.; NEAMTIU, I. Automating gui testing for android applications. In: ACM. *Proceedings of the 6th International Workshop on Automation of Software Test*. [S.l.], 2011. p. 77–83. Citado na página 21.

ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. [S.l.]: ISO/IEC, 2001. Citado na página 17.

MATHUR, A. P. *Foundations of Software Testing, 2/e*. [S.l.]: Pearson Education India, 2013. Citado na página 19.

NUNES, W. S. Implantação de práticas de integração contínua: um relato de experiência em um laboratório de pesquisa e desenvolvimento de software. 2017. Citado na página 14.

PEREIRA, I. M.; CARNEIRO, T. G. S.; PEREIRA, R. R. Developing innovative software in brazilian public universities: tailoring agile processes to the reality of research and development laboratories. In: *Proceedings of the 4th Annual Conference on Software Engineering and Applications (SEA 2013), Thailand, Bangkok, Global Science and Technology Forum (GSTF)*. [S.l.: s.n.], 2013. Citado na página 14.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016. Citado 2 vezes nas páginas 17 e 19.

REZENDE, D. A. *Engenharia de software e sistemas de informação*. [S.l.]: Brasport, 2005. Citado na página 15.

RODRIGUES, N. N.; ESTRELA, N. V. Simple way: Ensino e aprendizagem de engenharia de software aplicada através de ambiente e projetos reais. *Anais do VIII Simpósio Brasileiro de Sistemas de Informação,(São Paulo, 2012)*, 2012. Citado na página 14.

SHAH, S. M. A. et al. Towards a hybrid testing process unifying exploratory testing and scripted testing. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 26, n. 2, p. 220–250, 2014. Citado 2 vezes nas páginas 20 e 33.

SOMMERVILLE, I. *Software engineering*. 9th ed. ed. Pearson, 2011. ISBN 9780137035151,0137035152,0137053460,9780137053469. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=CA148588F264BE83BEE5DF58F9BB10E6>. Citado 2 vezes nas páginas 14 e 18.