Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

JOEL RODRIGUES ÁLVARES LEAL

PLUGINS DE APOIO AO TESTE DE MUTAÇÃO EM APLICAÇÕES MÓVEIS

Assis
2015

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302-1055 homepage: www.fema.edu.br

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

JOEL RODRIGUES ÁLVARES LEAL

PLUGINS DE APOIO AO TESTE DE MUTAÇÃO EM APLICAÇÕES MÓVEIS

Trabalho de Conclusão de Curso apresentado ao
Instituto Municipal de Ensino Superior de Assis,
como requisito de Curso de Graduação.

Assis
2015

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302-1055 homepage: www.fema.edu.br

FICHA CATALOGRÁFICA

LEAL, Joel Rodrigues Alvares

Plugins de Apoio ao Teste de Mutação em Aplicações Móveis/Joel Rodrigues Alvares Leal. Fundação Educacional do Município de Assis – FEMA: Assis, 2015.

102p.

Orientador: Prof. Esp. Guilherme de Cleva Farto

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis

1. Teste de Mutação **2.** Google Android **3.** Eclipse IDE **4.** PDE

CDD: 001.6

Biblioteca da FEMA

PLUGINS DE APOIO AO TESTE DE MUTAÇÃO EM APLICAÇÕES MÓVEIS

JOEL RODRIGUES ÁLVARES LEAL

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Prof. Esp. Guilherme de Cleva Farto
Analisador(1): Prof. Dr. Alex Sandro de Souza Romeo Poletto

Assis
2015

DEDICATÓRIA

Dedico este trabalho à Deus, a toda minha família
e principalmente meus pais por terem me dado todo amor,
carinho, educação e serem exemplos de pessoas para
eu poder ser a pessoa que sou hoje.
Dedico também a minha namorada, meu primo e meus
amigos que estiveram do meu lado sempre me
apoiando para realização deste sonho.

AGRADECIMENTOS

Agradeço primeiramente e **Deus** por ter me dado muita força e por sempre estar presente em minha vida, caminhando ao meu lado todos os dias, me fazendo mais forte e motivado.

A minha professora Dr **Marisa Atsuko Nitto**, por desde o primeiro ano me orientar em todo processo acadêmico e pelo ano de 2013 o qual me orientou em um Programa de Iniciação Científica.

Ao professor Dr **Almir Rogério Camilesi**, por desde o primeiro ano me orientar em todo processo acadêmico e pelo ano de 2014 e 2015 o qual me orientou e está me orientando em um Programa de Iniciação Científica.

Ao professor Esp. **Guilherme de Cleva Farto**, além de um excelente orientador, um amigo que me apoiou e me ajudou no decorrer do projeto transmitindo todo o conhecimento possível. Saiba que com você aprendi muito, existem muitas pessoas que podem fazer o mesmo, porém, poucas com grande dedicação e excelência.

A minha namorada **Juliana Barroso Lomiler** pelo apoio, ajuda, compreensão, incentivo ao decorrer do projeto.

A minha família **Paulo Cesar, Nivea e João Pedro** pelo apoio e incentivo. Obrigado pelas orações.

A todos meus **amigos e colegas de sala**, pelo apoio, paciência, amizade e companheirismo.

E por fim agradeço a **todas as pessoas** que colaboraram direta ou indiretamente para a execução deste trabalho.

“Que os vossos esforços desafiem as impossibilidades, lembrai-vos de que as grandes coisas do homem foram conquistadas do que parecia impossível.”

Charles Chaplin

RESUMO

O teste de mutação é uma das mais promissoras tecnologias da engenharia de software que envolve a etapa de testes, tem sido um dos temas mais estudado ultimamente em relação a testes. Em alguns locais o teste de mutação já é aplicado e vem sendo muito estudado e pesquisado para utilização em todos os meios de desenvolvimento. A proposta deste trabalho é de pesquisar e compreender os conceitos da tecnologia *Google Android, Plugin Development Environment (PDE)* e Teste de Mutação, assim desenvolvendo um *plugin* para Eclipse IDE que realize teste de mutação em aplicativos desenvolvidos na plataforma *Google Android* a fim de encontra o maior número de defeitos.

Palavras-chave: Teste de Mutação; Google Android; Eclipse IDE; PDE.

ABSTRACT

The mutation test is one of the most promising technologies of software engineering that involves the step of testing has been one of the subjects most widely studied lately regarding the tests. In some places the mutation test is already applied and it has been studied and researched for use in all development means. The purpose of this work is to research and understand the concepts of Google Android technology, *Plugin* Development Environment (PDE) and Mutation Testing, this developing a *plugin* for Eclipse IDE to conduct mutation assay applications developed in the Google Android platform the to find the largest number of defects.

Keywords: Mutation Testing; Google Android; Eclipse IDE; PDE.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Nos últimos anos, com o crescimento da área da tecnologia da informação, os *smartphones* tornaram-se um meio de comunicação muito utilizado por pessoas e empresas de pequeno, médio e grande porte. Segundo Deitel (2012), em Agosto de 2010, mais de 200.000 dispositivos *Android* foram ativados por dia, 100.000 a mais que dois meses antes. Em Junho de 2011, mais de 500.000 dispositivos *Android* foram ativados por dia. Tendo em vista o grande número de dispositivos com a tecnologia *Google Android*, consequentemente o número de aplicações desenvolvidas também cresceu com o objetivo de atender um maior número de usuários.

Neste cenário, o crescimento do desenvolvimento de aplicativos para esses aparelhos também aumentou, principalmente pela portabilidade e mobilidade que é dada pelos dispositivos móveis, possibilitando seu uso a qualquer momento e local. (TONIN, 2012).

A partir da grande demanda de desenvolvimento de aplicativos para tais aparelhos, softwares devem ser desenvolvidos para verificar o sistema implementado, pois uma das etapas mais demoradas e mais caras do desenvolvimento de um software é a etapa de teste (FALBO, 2005).

Existem várias maneiras de testar um software, uma delas é o teste em fases, que utiliza-se da estratégia “dividir para conquistar” e em cada etapa é realizado um tipo de teste focando em determinada característica do software, iniciando nas unidades menores, posteriormente na integração dessas unidades e incrementalmente, alcançando todo o sistema. Tais atividades devem ser desenvolvidas ao longo desenvolvimento do software, e em geral concretizam-se em três etapas, de unidade, de integração e de sistema. O teste de unidade concentra esforços na menor unidade do projeto, ou seja, pesquisa erros de lógicas e de implementação separadamente em cada módulo do sistema. O teste de integração é um exercício sistemático aplicado durante a integração da estrutura do programa, visando descobrir erros associados às interfaces entre os módulos, o objetivo é que a partir

do teste de unidade, seja possível construir a estrutura do programa determinada pelo projeto. O teste de sistema é realizado após o teste de integração, porque visa identificar erros de funções e desempenho que não estejam de acordo com as especificações do projeto (DEUS, 2009).

É fundamental o desenvolvimento de ferramentas de teste para o suporte à atividade de teste propriamente dita, uma vez que essa atividade é muito propensa a erros, além de improdutiva quando aplicada manualmente. Assim, a disponibilidade de ferramentas de teste proporciona maior qualidade e produtividade para as atividades de teste no decorrer do desenvolvimento de software (BARBOSA et al., 2000).

A análise de mutantes é uma técnica de teste baseada em erros que utiliza informações sobre erros mais frequentes no processo de desenvolvimento de software. A ênfase desta técnica está nos erros que o programador ou projetista pode cometer no processo de desenvolvimento (BARBOSA et al., 2000).

Os programadores experientes elaboram programas corretos ou o mais próximo do correto. Assumindo essa hipótese, pode-se afirmar que erros são introduzidos nos programas através de desvios sintáticos, que embora não causam erros sintáticos, alteram a semântica do programa, e consequentemente levam o programa a comportamento incorreto. Para descobrir tais erros, a análise de mutantes identifica os desvios sintáticos mais comuns e, através da aplicação de pequenas transformações sobre o programa em teste, encoraja o testador a construir casos de testes que mostrem que tais transformações levam a um programa incorreto (BARBOSA et al., 2000).

Outra hipótese explorada no critério da análise de mutantes é o efeito de acoplamento, o qual assume que erros complexos estão relacionados a erros simples. Sendo assim, alguns estudos já confirmam está hipótese, que conjuntos de casos de teste capazes de revelar erros simples, também são capazes de revelar erros complexos (BARBOSA et al., 2000).

Segundo Dantas (2009), os testes de aplicações móveis são muito mais complexos do que os testes de aplicações *desktop* ou *web* por conta das limitações encontradas em ambientes móveis como pouca memória, pouca capacidade de

armazenamento, baixo processamento, tela pequena e consumo de energia, limitações que não preocupam em desenvolvimento *desktop* ou *web*.

Segundo Nobre et al. (2012), o teste de mutação tem apresentado muita eficiência em termos de erros encontrados, mas um alto custo computacional, pelo fato da execução de um grande número de mutantes.

1.1 OBJETIVOS

O objetivo geral deste trabalho é o de pesquisar os conceitos de teste de mutação, bem como as ferramentas e arquiteturas necessárias para o desenvolvimento de componentes que contribuirão com a técnica de teste proposta para a verificação de aplicações desenvolvidas na plataforma *Android*.

A partir dos artefatos propostos, será possível aplicar a técnica de mutação e operadores de mutação em um ambiente de desenvolvimento de aplicações móveis *Android,* com a finalidade de gerar mutantes e contribuir com a identificação de casos de teste defeituosos.

1.1.1 Objetivos Específicos

Pretende-se com este trabalho, implementar componentes baseados em conceitos do teste de mutação para o ambiente de desenvolvimento Eclipse. De forma a tornar possível a elaboração e execução, tanto das etapas teóricas quanto das práticas, os seguintes objetivos específicos foram instituídos:

- Pesquisar a técnica de teste de mutação, operadores de mutação e mutantes*;*
- Pesquisar e analisar os métodos e algoritmos, bem como os operadores de mutação existentes;
- Pesquisar e analisar a plataforma *Google Android;*

- Pesquisar e analisar o desenvolvimento de componentes para o ambiente Eclipse;
- Especificar um estudo de caso:
    - Modelar o problema;
    - Desenvolver a arquitetura e documentação dos componentes*;*
    - Implementar o conjunto de componentes baseados nos conceitos de teste de mutação;
    - Definir um estudo de caso;
    - Testar e validar o conjunto de componentes propostos;
    - Descrever os resultados obtidos.

1.2 JUSTIFICATIVAS

Tendo em vista o grande número de *smartphones* vendidos, o número de aplicativos desenvolvidos para plataforma *Google Android* cresce paralelamente, aplicações de entretenimento, comerciais, jogos, mapas, rádio, saúde, *chats,* entre outros tipos de softwares. Quanto mais sistemas são desenvolvidos, mais propícios a erros estão e, com isso, é necessário uma ferramenta que auxilie no teste para garantir uma maior qualidade do software (EXAME, 2014).

1.3 MOTIVAÇÃO

O desenvolvimento deste trabalho de pesquisa consiste no fato de que o teste de mutação é um tema ainda pouco explorado e pode contribuir com a qualidade da aplicação testada.

O teste de mutação ajuda a manter a qualidade das aplicações desenvolvidas, tais testes podem ser utilizados em diversos domínios, como por exemplo, aplicações de entretenimento, comerciais, jogos, mapas, rádio, *chats,* entre outros tipos de softwares.

Aplicar os conceitos estudados para o desenvolvimento do componente na plataforma *Google Android* agregará muito valor aos desenvolvedores, pois não precisarão testar suas aplicações manualmente. Eles terão um componente pronto e específico para testar seus aplicativos desenvolvidos, assim tornando mais rápido o seu desenvolvimento, visto que os testes não lhe ocuparão tempo algum.

Outra motivação é a chance de, num futuro não muito distante, o mercado de trabalho necessitar de profissionais com conhecimento na linha do tema desta pesquisa, uma vez que a área de desenvolvimento de aplicativos para dispositivos móveis cresce exponencialmente a cada dia.

1.4 ESTRUTURA DO TRABALHO

O trabalho está estruturado nas seguintes partes:

- **Capítulo 1 – Introdução**
- **Capítulo 2 – Teste de Software**
- **Capítulo 3 – Tecnologia *Google Android***
- **Capítulo 4 – Desenvolvimento de Componentes para Eclipse IDE**
- **Capítulo 5 – Proposta do Trabalho**
- **Capítulo 6 – Estudo de Caso**
- **Capítulo 7 - Conclusão**
- **Referências**

2. TESTE DE SOFTWARE

Neste capítulo serão apresentados os principais conceitos acerca do teste de software, bem como suas definições fundamentais e básicas para o seu entendimento. Além disso, o Teste de Mutação será apresentado.

2.1 INTRODUÇÃO

Considerar a qualidade de um software é um dos principais objetivos da engenharia de software que, mesmo com aprimoramento de métodos, ferramentas e técnicas de desenvolvimento de software, defeitos ainda podem existir. Para isso, atividades específicas auxiliam na detecção de defeitos em softwares, pois com os testes convencionais que são realizados, muitas vezes esses tipos de erros não são identificados por acontecer apenas quando tiver uma certa entrada de dados em particular, um valor especifico, em uma condição que entra apenas em casos específicos, entre outros erros introduzidos (ASCARI, 2009).

Em um sentido mais abrangente, o teste de software pode ser visto como uma atividade de verificação, validação e teste. A verificação e validação são consideradas atividades estáticas com o intuito de confirmar que o software cumpra todas suas especificações e atenda às necessidades do usuário. A verificação identifica a construção do produto, se está seguindo o processo de desenvolvimento e gerando as saídas desejadas. A validação diz respeito ao conforto que o usuário irá ter com o produto construído, se realmente foi o produto almejado por ele (CAMPANHA, 2010).

O objetivo da atividade de teste de *software*, ao contrário do que pode parecer, não é garantir que o programa não tenha falhas, mas sim encontrar o maior número de falhas possíveis. Portanto, um teste é considerado eficiente quando alguma falha é encontrada (CAMPANHA, 2010).

É de extrema importância conhecer a nomenclatura utilizada na diferenciação dos termos de “defeito”, “engano”, “erro” e “falha”:

- Defeito *(fault)* – passo, processo ou definição de dados incorretos, é considerado um defeito uma instrução ou comando escrito incorreto.
- Falha *(failure)* – saída de dados incorreta em relação a que foi especificado;
- Erro *(error)* – diferença entre o valor esperado e o valor obtido, ou seja, um resultado incorreto, inesperado na execução do programa;
- Engano *(mistake)* – ação humana que provocou o resultado incorreto (defeito), como por exemplo, uma ação incorreta tomada pelo programador;

2.2 TIPOS E TÉCNICAS DE TESTE DE SOFTWARE

Para melhor gerenciamento dos testes de software, recomenda-se sua aplicação em etapas. Segundo Pressman (2006), as fases ou etapas propostas são assim definidas:

1. Teste de unidade: Corresponde à verificação da menor unidade de software, o módulo; visa identificar erros de lógica e implementação separadamente;
2. Teste de integração: Valida a integração entre os módulos do software, busca erros de interface entre os mesmos;
3. Teste de validação: Verifica se o produto está de acordo com os requisitos do software informado pelo cliente. É idealizado a partir da execução de uma série de teste funcional. Os resultados dos testes demonstram a conformidade aos requisitos.
4. Teste de sistema: É uma serie de diferentes testes com a finalidade de exercitar o software por completo. Apesar de composto por diferentes testes específicos, são todos exercitados para verificar se toda integração do software está trabalhando de forma correta e executando as funções para eles alocadas.

Independente da fase de teste que está sendo aplicada, há etapas bem definidas para execução de tal atividade. Em Sommerviller (2003), são descritas as quatro principais etapas da atividade de teste:

- Planejamento de Testes: onde são definidos os testes que serão aplicados, de acordo com os recursos disponíveis para o testador, será planejado o cronograma e as responsabilidades de cada pessoa durante os testes. É nesta fase que são definidas as ferramentas, objetivos e métodos;
- Projeto dos Casos de Teste: onde é projetada uma estratégia para a construção dos casos de teste com objetivo de alcançar as condições dos critérios estabelecidos;
- Execução do Teste: após a execução das fases anteriormente descritas, as saídas dos testes são comparadas com a saída esperadas, caso um erro é encontrado, esses são documentados para ser posteriormente corrigidos;
- Coleta e avaliação dos resultados dos testes: os resultados dos testes são registrados, organizados e apresentados como relatório.

Além dos conceitos básicos de teste de software é de mera importância conhecer os conceitos básicos e as principais técnicas existentes. Os critérios de teste servem para auxiliar o testador na construção dos casos de testes e também pode fornecer uma medida de qualidade dos conjuntos de CT construídos de acordo com os requisitos estabelecidos para cada critério (CAMPANHA, 2010).

De acordo com a etapa do teste do software e ainda o contexto em que o software está sendo projetado e desenvolvido, o testador pode decidir entre várias técnicas, a que deseja aplicar. Todas as técnicas têm o mesmo objetivo: encontrar defeitos em um software (ASCARI, 2009).

2.1.1 Teste Funcional

São gerados diversos artefatos incluindo o próprio software, isso a partir das próprias especificações. A técnica funcional é voltada para a elaboração de casos de testes a partir de informações contidas nos requisitos do software que está sendo fabricado (BEIZER, 1990; MYERS et al. 2004).

Teste funcional, também conhecido como teste de caixa preta, essa técnica é baseada nas especificações do software, as únicas informações que ela conhece do software são: 1) especificação; 2) entrada de dados do software; e 3) saídas esperadas. Com essa técnica, são criados a partir dos requisitos os casos de testes que basicamente contém as entradas de informações do software e as saídas esperadas que o software produza. A tarefa do testador é dirigir o sistema, como se não conhecesse o funcionamento interno e analisar se o que é gerado corresponde ao esperado pelas especificações (DEUS, 2009).

Segundo Deus (2009), alguns passos que podem ser utilizados para realização do teste de software utilizando a técnica de teste funcional:

1. Analisar a especificação de requisitos;
2. Identificar a partir da especificação entradas válidas e inválidas para os sistemas;
3. Determinar as saídas esperadas para cada conjunto de entrada;
4. Produzir e documentar o caso de teste;
5. Executar o caso de teste conformar projetado;
6. Capturar as saídas produzidas pelo software e comparar com as saídas esperadas;
7. Documentar a comparação entre o resultado esperado e o produzido.

Lembrando que esse tipo de teste pode ser realizado em qualquer fase do desenvolvimento, ou seja, pode ser aplicado no teste de unidade, teste de integração, teste de validação ou teste de sistema, e quando o cliente receber o produto final o mesmo pode utilizar esses testes para avaliar o sistema, aceitando ou

rejeitando o produto. E também, para que os testes tenham eficiência os requisitos do sistema têm que ter sido feito com muita qualidade, senão corre risco do software apresentar defeitos nos testes em módulos que estão corretos, tudo isso por conta de especificações incorretas, como citada anteriormente, os testes são realizados a partir das especificações do software (DEUS, 2009).

Segundo Fabbri et al. (2007), os principais critérios que pertence a técnica de teste funcional são descrito a seguir:

- Particionamento em Classes de Equivalência: com base nas funcionalidades descritas no documento de especificação, os possíveis valores do domínio de entrada são divididos em classes representativas. Tais classes são separadas em válidas e inválidas dependendo da aceitação ou rejeição dos dados pelo programa testado. Esse critério é baseado na hipótese de que um elemento de uma classe representa o comportamento de qualquer outro elemento dessa mesma classe.

  Para o critério de particionamento em classe de equivalência seja satisfatório, a condição é que o conjunto de casos de teste satisfaça todas as classes de equivalência com pelo menos um caso de teste para cada classe inválida.
- Análise do Valor Limite: Esse critério de teste busca para determinada entrada avaliar os valores que estão nas fronteiras superiores e inferiores ao próprio limite do software. Por exemplo, se um determinado software existe um procedimento de entrada válido entre 0 e 100, utilizando esse critério deveriam ser criados testes para os valores 99, 100 e 101, ou seja, uma unidade exatamente abaixo, o próprio valor de limite e uma unidade exatamente acima do valor de fronteira. De acordo com essas características, é possível determinar a precisão do software.
- Teste Funcional Sistemático *(Systematic Functional Testing):* Segundo Linkman et al. 2003, esse teste é a união dos dois critérios funcionais citados anteriormente, ou seja, a junção do Particionamento em Classes de Equivalência com a Análise do Valor Limite.

2.2.2 Teste Estrutural

Segundo Deus (2009), o teste estrutural ou teste caixa branca define seus requisitos com base nas informações codificadas, tem o intuito de verificar a estrutura dos comandos. Quando um comando não é executado pelo menos uma vez, não [e possível garantir que ele está isento de defeito.

Um software é composto por vários algoritmos, esses são compostos por vários comandos condicionais, laços de repetições, declarações e inicialização de variáveis, computação de valores, entre outros. Quando um engenheiro projeta um algoritmo que nuca é utilizada pelo software ou um escopo que nunca é executado, dificilmente o cliente fica sabendo disso, porém, quando um membro de sua equipe começa a manutenção no software ele descobre esse problema. Isso acontece quando não é utilizado o teste estrutural (DEUS, 2009).

O testador quando realiza esse teste tem que conhecer a estrutura do sistema por completo, estrutura interna incluindo o código fonte (ASCARI, 2009).

Executando o teste estrutural é possível afirmar que se o teste cobrir 100% do software, dificilmente irá ocorrer posteriores problemas e sim, apenas algumas manutenções, caso esse teste tenha coberto 50% do software, pode-se garantir que de todos os testes realizados desde o de unidade, integração, validação e sistema, apenas 50% do software está funcionando perfeitamente.

2.2.3 Outros Tipos de Teste de Software

Segundo Ascari (2009), a técnica do teste de software baseada em defeitos, tem como base os tipos de defeitos mais frequentes no processo do desenvolvimento de um determinado software para originar os casos de teste que podem ser utilizados para identificar esses defeitos.

Os tipos de defeito que um desenvolvedor pode cometer são definidos pelas formas semelhantes utilizadas para expressar o que tem na sua mente, como a forma que ele utiliza para desenvolver (BINDER, 1999).

Sendo assim, a linguagem, método ou técnica utilizada para o desenvolvimento são de base para as informações coletadas para realização da técnica baseada em defeitos, para definir os tipos de erros que estarão presentes no software. A semeadura de Defeitos e a Análise de Mutantes são metodologia que se sobressaem nessa técnica de teste de software (ASCARI, 2009).

Segundo Deus (2009), a técnica *ad hoc* também conhecida como teste aleatório ou sem critério especificado, provavelmente a mais utilizada pelos engenheiros de software que não focam outras técnicas de teste. Tal técnica não é especifica para determinada parte do programa, quem escolhe o que observar é o próprio testador, se ele quiser observar o código fonte, realizar o teste funcional e avaliar os resultados junto às especificações ele pode fazer. Ele pode mesclar várias técnicas que julgue essencial para revelar o maior número de defeitos possíveis, ou seja, o testador pode fazer o que ele quiser para encontrar defeitos. Como vantagem, é possível dizer que essa técnica é a mais fácil para dar treinamento, por ser mais simples de aprender e não existem ferramentas complexas para ela. Como desvantagem a dependência do conhecimento do testador, a qualidade do teste fica totalmente dependente de sua criatividade em revelar defeitos, ele só para de testar quando achar que já testou o suficiente e pode confiar no software testado.

2.3 TESTE DE MUTAÇÃO

Nesta seção, será apresentado o teste de mutação, destacando seus principais conceitos, objetivos e utilização no teste de software.

2.3.1 Introdução

No teste de mutação, também conhecido como análise de mutantes, são introduzidos pequenos desvios sintáticos sobre o código fonte, embora não causam erros sintáticos, alteram a semântica do programa e consequentemente levam o programa ao erro. Esses programas pouco alterados são chamados de mutantes e o objetivo desse teste é gerar casos que a partir da mesma entrada os resultados gerados pelo programa original seja diferente dos resultados gerados pelo programa mutante (DEMILLO et al., 1978).

Para realizar essas modificações nos programas mutantes são utilizados operadores de mutação. Quando um operador de mutação aplicado em um programa P, transforma P em um programa M, ou seja, um programa mutante com apenas uma alteração em um operador. Além de modelar alguns defeitos simples cometidos por programadores, os operadores de mutação podem ser determinados para requerer algumas características especificas dos casos de testes. Cada operador de mutação pode ser aplicado em várias partes do software, porém isso pode ser custoso por que a quantidade de mutantes gerados será muito grande (CAMPANHA, 2010).

Outra hipótese estudada na aplicação do teste de mutação é o efeito de acoplamento, o qual se assume que erros complexos estão relacionados a erros simples, sendo assim, testes capazes de revelar erros simples também são capazes de revelar erros complexos (DEMILLO et al., 1978).

A primeira etapa do teste de mutação é a geração de mutantes “M” a partir dos operadores de mutação escolhidos pelo testador. Após esta etapa, um conjunto de casos de testes “T” deve ser definido e executado cada caso de teste “t” com o programa original “P”, verificando o comportamento do programa. Caso os resultados gerados sejam diferentes dos resultados especificados então um erro foi encontrado, caso contrário, são executados cada mutante “m” com cada caso de teste “t” pertencente aos casos de teste. Se as saídas geradas forem diferentes das saídas especificadas então o mutante é considerado morto, caso as saídas forem iguais o mutante “m” está vivo, podendo ser equivalente ao programa “P”, sendo

esse impossível de ser morto pelos casos de teste, ou seja, “m” e “P” com os casos de teste “T” computam a mesma função (ASCARI, 2009; CAMPANHA, 2010).

Segundo DeMillo e Offutt (1991), para um caso de teste t pertencente ao conjunto de casos de testes T mate um mutante m pertencente ao conjunto de mutantes gerados M em relação ao programa P, três condições estão associadas:

- Alcançabilidade: O elemento que sofreu mudança no programa m deve ser executado pelo caso de teste t;
- Necessidade: Após a execução da mutação o estado do mutante m deve diferir do estado do programa p no mesmo momento da execução;
- Suficiência: O resultado da execução de m deve diferir do resultado da execução de P, ou seja, a mutação executada deve refletir até o encerramento da execução do mutante m e do programa P.

A Figura 1 ilustra o uso do teste de mutação.

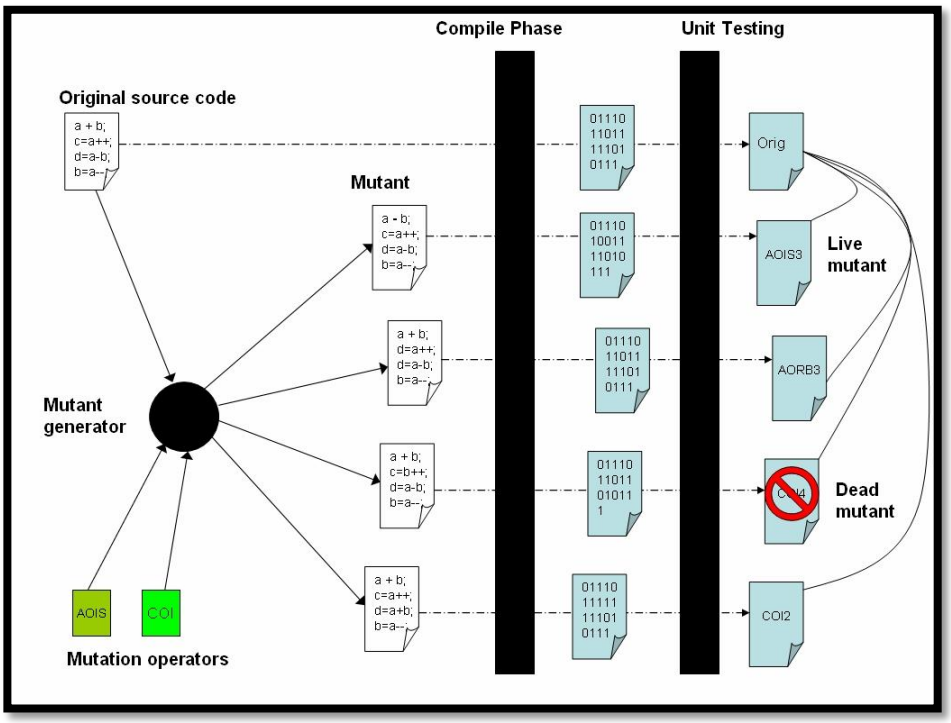

Figura 1 – Exemplo do uso do teste de mutação (ENDEL9, 2015).

2.3.2 Operadores de Mutação e Exemplos de Uso

No desenvolvimento de operadores de mutação a estratégia utilizada é utilizar todas as possíveis mudanças sintáticas para recursos de orientação a objetos.

Geralmente, todos os operadores de mutação entram em três categorias: (1) excluir, (2) inserir e, (3) alterar um elemento sintático-alvo.

Operadores de mutação para ser utilizado no encapsulamento, descrito por MA e OFFUTT, 2014.

Sigla: AMC

Título: *Access modifier change*

Descrição: Operador de mutação responsável pela mudança no modificar de acesso.

Este operador de mutação consiste em modificar o operador de acesso com a intenção de verificar se os atributos estão sendo acessados de maneira correta.

A Figura 2 ilustra um exemplo de AMC.

```
//Programa Original
public long ra;

//Programa Mutante
private long ra;
```

Figura 2 - Código fonte exemplificando AMC.

Operadores de mutação para ser utilizado na herança, descrito por MA e OFFUTT, 2014.

Sigla: IHD

Título: *Hiding variable deletion*

Descrição: Operador de mutação responsável por eliminar atributos de subclasses com mesmo nome e tipo de um atributo da classe pai.

Este operador de mutação consiste em eliminar um ou mais atributo de subclasses cujo nome e tipo são idênticos ao da classe pai, com isso é realizado testes para verificar se a manipulação dos atributos está sendo realizada de forma correta.

A Figura 3 ilustra um exemplo de IHD.

```
//Programa Original
class Pessoa {
      int cod;
      ...;
 }
class Professor extends Pessoa {
      int cod;
      ...;
}

//Programa Mutante
class Pessoa {
      int cod;
      ...;
 }
class Professor extends Pessoa {
      //int cod;
      ...;
}
```

Figura 3 - Código fonte exemplificando IHD.

Sigla: IHI

Título: *Hiding variable insertion*

Descrição: Operador de mutação responsável por inserir atributos em subclasses com o mesmo nome e tipo de um atributo da classe pai.

Este operador de mutação consiste em inserir um atributo na subclasse cujo nome e tipo são idênticos ao da classe pai, com isso é realizado testes para verificar se a manipulação dos atributos da classe pai está sendo realizada de maneira correta.

A Figura 4 ilustra um exemplo de IHD.

```
//Programa Original
Original
"class Pessoa {
      int cod;
      ...;
}
class Professor extends Pessoa {
      ...;
}

//Programa Mutante
class Pessoa {
      int cod;
      ...;
}
class Professor extends Pessoa {
      int cod;
      ...;
}
```

Figura 4 - Código fonte exemplificando IHI.

Sigla: IOD

Título: *Overriding method deletion*

Descrição: Operador de mutação responsável por deletar métodos de subclasses, sendo assim quando o método é invocado, o método utilizado é a versão da classe pai.

Este operador de mutação consiste em eliminar um método da subclasse cujo nome, parâmetros e retorno sejam idênticos à um método da classe pai, com isso quando o método for invocado, o método utilizado será o da classe pai, com isso é possível verificar se os métodos realizam a mesma computação e são equivalentes, ou caso

contrário qual seria a segurança aplicada para a não invocação do método da classe pai.

A Figura 5 ilustra um exemplo de IOD.

```
//Programa Original
class Professor extends Pessoa {
	...;
	void salario (int n1) {
		...;
	}
}

//Programa Mutante
class Professor extends Pessoa {
	...;
	//void salario (int n1) {...;}
}
```

Figura 5 – Código fonte exemplificando IOD.

Sigla: ISI

Título: *Super keyword insertion*

Descrição: Operador de mutação responsável por inserir a palavra *super*, a fim de que uma referência para o atributo ou método vai para a classe pai, esse teste foi projetado para verificar se as atributos estão sendo utilizadas em lugares corretos.

A Figura 6 ilustra um exemplo de ISI.

```
//Programa Original
class Professor extends Pessoa {
	...;
	int salario () {
		...;
		return val*num;
	}
}

//Programa Mutante
class Professor extends Pessoa {
	...;
	int salario () {
```

```
            ...;
            return val*super.num;
      }
}
```

Figura 6 – Código fonte exemplificando ISI.

Sigla: IPC

Título: *Explicit call to a parente`s constructor deletion*

Descrição: Operador de mutação responsável por excluir chamada de construtores parametrizados de uma subclasse.

Este operador de mutação consiste em eliminar uma chamada de construtor parametrizado, sendo assim o construtor chamado é o padrão, com isso é possível verificar qual problema ocorre caso o construtor parametrizado não for invocado, assim, é possível tratar este erro.

A Figura 7 ilustra um exemplo de IPC.

```
//Programa Original
class Professor extends Pessoa {
      ...;
      Professor (int a) {
            super(a);
            ...;
      }
}

//Programa Mutante
class Professor extends Pessoa {
      ...;
      Professor (int a) {
            //super(a);
            ...;
      }
}
```

Figura 7 – Código fonte exemplificando IPC

Operadores de mutação para ser utilizado no polimosfismo, descrito por MA e OFFUTT, 2014.

Sigla: PNC

Título: *New method call with chils class type*

Descrição: Operador de mutação responsável por alterar o tipo instanciado de um objeto.

Este operador de mutação consiste em alterar o tipo instanciado de um objeto com o intuito de verificar se a manipulação dos atributos e métodos está sendo realizado de maneira correta.

A Figura 8 ilustra um exemplo de PBC, e na Figura a classe *Parent* é pai da classe *Child*.

```
//Programa Original
Parent a;
a = new Parent();

//Programa Mutante
Parent a;
a = new Child();”
```

Figura 8 – Código fonte exemplificando PNC

Sigla: PMD

Título: *Member variable declaration with parent class type*

Descrição: Operador de mutação responsável por alterar o tipo da referência declarada de um objeto para o tipo da classe pai.

Este operador de mutação consiste em alterar o tipo da referência de um objeto com o intuito de verificar se a manipulação dos atributos e métodos está sendo realizada de maneira correta.

A Figura 9 ilustra um exemplo de PNC, e no exemplo a classe *Parent* é pai da classe *Child*.

```
//Programa Original
Child a;
a = new Child();

//Programa Mutante
Parent a;
a= new Child();
```

Figura 9 – Código fonte exemplificando PNC

Sigla: PPD

Título: *Parameter variable declaration with child class type*

Descrição: Operador de mutação responsável por alterar o tipo dos atributos que passam no parâmetro. Ele muda o tipo declarado do objeto para o tipo da classe pai.

Este operador de mutação consiste em alterar o tipo dos atributos ou objetos que passam por parâmetro, assim verificando se a manipulação dos atributos e dos objetos está sendo realizada de maneira correta.

A Figura 10 ilustra um exemplo de PPD, e no exemplo a classe *Parent* é pai da classe *Child*.

```
//Programa Original
boolean equals (Child o) {...}

//Programa Muante
boolean equals (Parent o) {...}
```

Figura 10 – Código fonte exemplificando PPD

Sigla: OMD

Título: *Overloading method deletion*

Descrição: Operador de mutação responsável por deletar sobrecarga de método, uma de cada vez.

Este operador de mutação consiste em eliminar uma sobrecarga de método, afim de verificar se os métodos estão sendo invocados de maneira correta.

A Figura 11 ilustra um exemplo de OMD:

```
//Programa Original

class List {
      ...;
      ...;
      void Add (int e) {... ...}
      void Add (float n) {... ...}
}

//Programa Mutante

class List {
      ...;
      ...;

      //void Add (int e) {... ...}
      void Add (float n) {... ...}
}
```

Figura 11 – Código fonte exemplificando OMD.

Sigla: OAC

Título: *Argument of overloading method change*

Descrição: Operador de mutação responsável por alterar a ordem ou o número de argumentos em chamadas de métodos, isso acontece apenas se houver uma sobrecarga de método que pode aceitar novas listas de argumento.

Este operador de mutação consiste em alterar a ordem dos atributos passados para o método, assim verificando se os métodos estão sendo utilizados de maneira correta e os atributos passados na ordem correta.

A Figura 12 ilustra um exemplo de OAC:

```
//Programa Original
p.Push(0.5, 2);

//Programa Mutante
p.Push(2, 0.5);
p.Push(2);”
p.Push(0.5);”
p.Push();”
```

Figura 12 – Código fonte exemplificando OAC.

Operadores de mutação para ser utilizado em operadores aritméticos, descrito por MA e OFFUTT, 2011.

A linguagem de programação Java suporta cinco operadores aritméticos, sendo eles para números inteiros e pontos flutuantes (*float),* os operadores são todos binários, sendo eles +, -, *, / e %. No entanto + e – tem versões unárias sendo definidas por quatro atalhos mt++, ++mt, mt-- e –mt.

1. AOR*B*: *Arithmetic Operator Replacement* – Substitui operadores binários aritméticos básicos por outros operadores binários aritméticos.
2. AOR*U*: *Arithmetic Operator Replacement* – Substitui operadores unários aritméticos básicos por outros operadores unários aritméticos.
3. AOR*S*: *Arithmetic Operator Replacement* – Substitui operadores binários aritméticos de atalhos por outros operadores binários aritméticos básicos.
4. AOI*U*: *Arithmetic Operator Insertion* – Insere operadores aritméticos unários básicos.
5. AOI*S*: *Arithmetic Operator Insertion* – Insere operadores aritméticos de atalhos.
6. AOD*U*: *Arithmetic Operator Deletion* – Deleta operadores aritméticos unários básicos.
7. AOD*S*: *Arithmetic Operator Deletion* – Deleta operadores aritméticos de atalhos.

Operador de mutação para ser utilizado em operador relacional, descrito por MA e OFFUTT, 2011.

Um operador relacional compara entre dois valores uma determinada relação entre eles. Java tem seis tipos de operadores, sendo eles <, <=, >, >=, != e ==

1. ROR: *Relational Operator Replacement* – Substitui operadores relacionais por outros operadores relacionais.

Operadores de mutação para ser utilizado em operadores condicionais, descrito por MA e OFFUTT, 2011.

A linguagem de programação Java possui suporte para seis operadores condicionais, cinco operadores binários e um operador unário. Os cinco operadores condicionais binários são &&, ||, &, | e ^. O operador condicional unário é “!”.

1. COR: *Conditional Operator Replacement* – Substitui operadores condicionais binários por outros operadores condicionais binários.
2. COI: *Conditional Operator Insertion* – Insere um operador condicional unário.
3. COD: *Conditional Operator Deletion* – Deleta um operador condicional unário.

Operador de mutação para ser utilizado em operador de deslocamento, descrito por MA e OFFUTT, 2011.

Java possui três operadores de deslocamento, sendo eles >>, << e >>>>. O operador de deslocamento realiza a manipulação dos bits de um dado, manipulando bit por bit para esquerda ou para direita.

1. SOR: *Shift Operator Replacemente*: Substitui um operador de deslocamento por outro operador de deslocamento.

Operadores de mutação para ser utilizado em operadores lógicos, descrito por MA e OFFUTT, 2011.

Java possui quatro operadores lógicos para executar funções bit a bit, três operadores lógicos são binários e um unário. Os três operadores lógicos binários são &, | e ˆ. O operador logico unário é ˜.

1. LOR*: Logical Operator Replacement* – Substitui operadores lógicos binários por outros operadores lógicos binários.
2. LOI: *Logical Operator Insertion* – Insere um operador logico unário.
3. LOD: *Logical Operator Delete* – Deleta um operador logico unário.

Operador de mutação para ser utilizado em operador de atribuição, descrito por MA e OFFUTT, 2011.

Java possui onze operadores de atribuição, o operador de atribuição básico atribui o valor da expressão com a variável do lado direito para a variável do lado esquerdo, os operadores de atribuição são +=, -=, *=, /=, %=, &=, |=, ˆ=, <<=, >>= e >>>=.

1. ASRS*s*: *Short-Cut Assignment Operator Replacement* – Substitui operadores de atribuição de atalho por outros operadores de atribuição de atalho.

2.3.3 Vantagens e Desvantagens

Um dos maiores problemas para o teste de mutação é o grande número de mutantes que é gerado, pois isso tem um alto custo computacional. Mesmo em pequenos programas o número de mutantes gerados pode ser muito grande, exigindo um tempo de execução muito alto. Várias estratégias estão sendo propostas para que o teste de mutação seja aplicado de forma mais eficiente. Uma

solução bastante explorada pela comunidade de teste é diminuir o número de mutantes a serem executados e analisados. Com essa ideia, algumas abordagens derivadas da análise de mutantes foram propostas: Mutação Aleatória *(Randomly Selected Mutation),* Mutação Restrita *(Constrained Mutation)* e Mutação Seletiva *(Selective Mutation).* Todas essas abordagens procuram selecionar um subconjunto do total de mutantes gerados, assim reduzindo o custo, mas com a expectativa de não reduzir a eficácia do teste (BARBOSA et al., 2000).

Outra característica que encarece a aplicação do teste de mutação é a identificação de mutantes equivalentes, um mutante equivalente é aquele que o programa original e o programa mutante, contra os casos de teste geram a mesma saída. A determinação de programas equivalentes deve ser feita manualmente pelo testador (CAMPANHA, 2010).

3. TECNOLOGIA *GOOGLE ANDROID*

Neste capítulo serão apresentadas as principais características e funcionalidades da tecnologia *Google Android,* bem como camadas da plataforma e os recursos de programação.

3.1 INTRODUÇÃO

Em 2014 a estimativa da União Internacional das Telecomunicações (UIT) é que até o final deste ano o número de celulares do mundo chegará a quase sete milhões, um número muito próximo ao número de habitantes no planeta. Até dezembro vai chegar a 96% o número de celulares no planeta, graças aos países emergentes que representarão 78% de todos os celulares espalhados por todo mundo (ENDEL1, 2014).

No Brasil, segundo os dados da Agência Nacional de Telecomunicações (Anatel), em Novembro de 2010 a quantidade de linhas ativas no Brasil passa de uma linha por habitante. Em Março de 2014, dados divulgados pela Anatel, o Brasil está com mais de 273,5 milhões de linhas móveis ativas (ENDEL1, 2014).

O que antigamente era luxo que tinha apenas em aparelhos de custo elevado, hoje qualquer usuário comum pode ter um em suas mãos, buscando cada dia jogos, câmeras de alta definição, músicas, *Bluetooth,* melhores jogos, GPS, acesso cada vez mais rápido com a Internet e principalmente estar conectado a redes sociais (LECHETA, 2013).

O mercado corporativo cresce e muitas empresas acoplam aplicações móveis no seu dia a dia, com o intuito de agilizar negócios e integrar aplicações móveis com seus sistemas legados. Todas as empresas visam lucros e, dessa forma os celulares podem ocupar um importante espaço em que a mobilidade está cada vez mais presente e o custo benefício de um dispositivo tem reduzido (LECHETA, 2013).

Quando lançado, o *Android* causou um grande impacto, chamando atenção de desenvolvedores e empresas. Pode-se dizer que isso aconteceu, pois uma empresa, a Google, está revolucionando a internet. No entanto, não é apenas a Google que está à frente disso, mas um grupo conhecido como Open *Handset Alliance (OHA),* um consórcio formado por empresas líderes no mercado de telefonia no mundo (LECHETA, 2013).

Dentro de tal plataforma as telas são chamadas de *activity*, quando se cria um mapa de objetos, são chamados de *bundle.* Tudo o que é desenvolvido, tem alterações no arquivo *AndroidManifest.xml,* tal arquivo é um dos mais importantes no desenvolvimento de aplicações *Google Android,* e dentro de tal arquivo existe versão mínima, máxima e recomendada para execução da aplicação, permissão para utilização do cartão de memória, câmera, dados móveis, entre outros. Todas as telas desenvolvidas, obrigatoriamente tem que ser listada em tal arquivo.

3.2 *OPEN HANDSET ALLIANCE (OHA)*

O grupo *Open Handset Alliance (OHA*) foi criado com a intenção de padronizar uma plataforma de código aberto e livre para celulares, com a necessidade de atender todas as expectativas e tendências do mercado. Esse grupo, formado por empresas referencias no mercado de dispositivos moveis, entre elas: HTC, LG, Motorola, Samsung, Sony Ericsson, China Mobile, T-Mobile, Intel, Toshiba, Strint Nextel, ASUS, Garmin e liderado pela poderosa Google tem o objetivo de mudar o que ocorre com o mercado com os dispositivos moveis, construindo uma moderna plataforma, única, aberta, moderna e flexível para o desenvolvimento de aplicações para qualquer tipo de usuário e corporações, assim resultando na tecnologia *Google Android* (FARIA, 2008).

Existe uma ótima descrição do que seria essa aliança no site da OHA. O texto está em inglês e a tradução da citação está aqui:

> Hoje, há um bilhão e meio de televisores em uso em todo o mundo. Um bilhão de pessoas está na Internet. Mas quase três bilhões de pessoas têm um telefone celular, e é um dos produtos de consumo mais bem sucedidas do mundo. Construir um dispositivo móvel melhor iria enriquecer a vida de inúmeras pessoas em todo o mundo. A Open Handset Alliance ™ é um grupo de líderes de móveis e tecnologia que compartilham esta visão para mudar a experiência móvel para os consumidores [...].

Assim, o objetivo do grupo é definir uma plataforma única e aberta para dispositivos móveis, assim, deixando os consumidores mais satisfeitos com o produto final. Outro objetivo principal é criar uma plataforma moderna e flexível para o desenvolvimento de aplicações corporativas. O resultado dessa união foi o nascimento do *Android* (LECHETA, 2013).

O *Android* é a nova plataforma de desenvolvimento para aplicativos móveis como *smartphones*, *tablets* e com um sistema operacional baseado em Linux, uma interface visual, GPS, várias aplicações já instaladas e um ambiente de desenvolvimento bastante poderoso, inovador e flexível. Também se pode utilizar a consagrada linguagem Java para desenvolvimento das aplicações, usufruindo de todos os recursos a que são disponibilizados (LECHETA, 2013).

3.3 PLATAFORMA *GOOGLE ANDROID*

A Figura 13 mostra toda arquitetura da plataforma *Android*. Logo em seguida será feita uma descrição detalhada de cada camada/nível da arquitetura, pois é de mera importância o entendimento de cada uma delas para o desenvolvimento de um aplicativo e principalmente para o desenvolvimento de *plugins* que testam os aplicativos desenvolvidos.

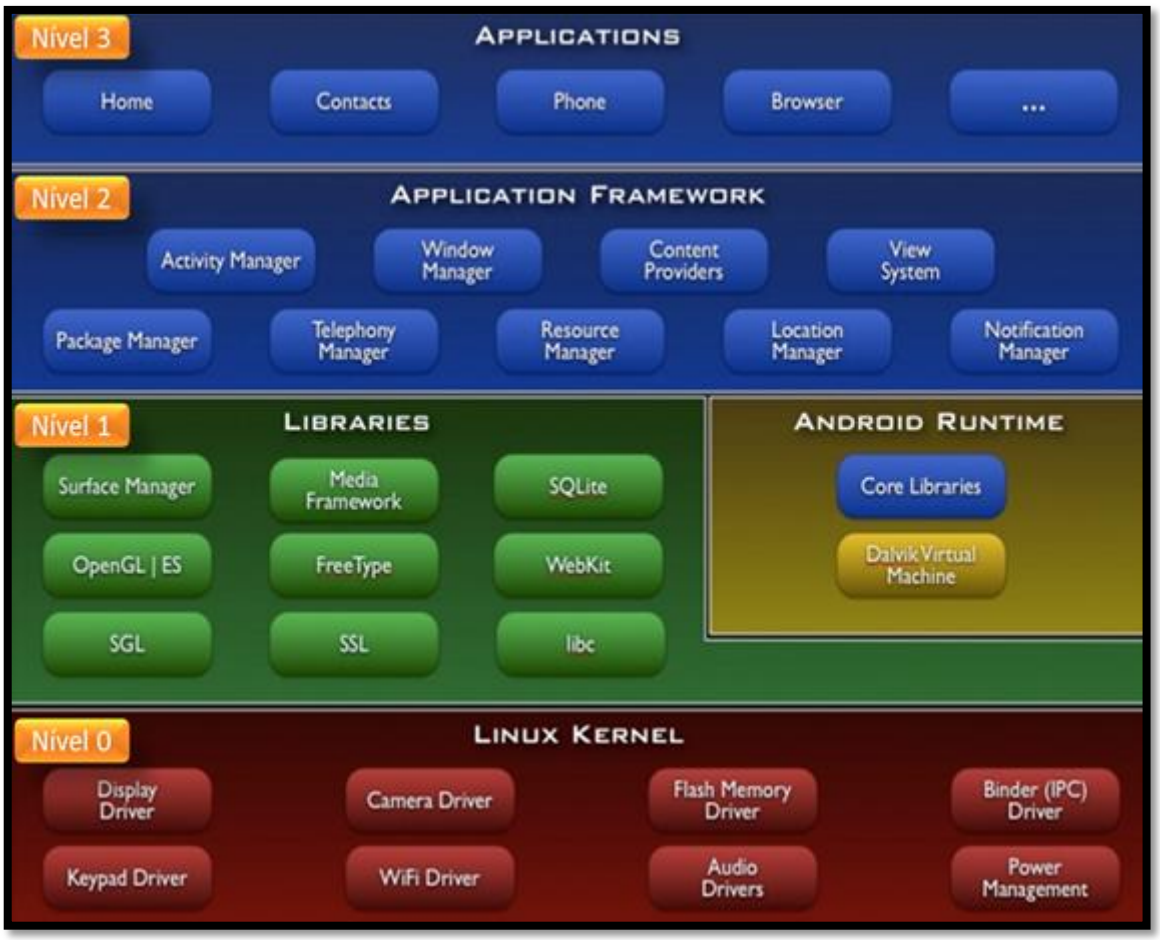

Figura 13 – Arquitetura *Google Android*

(In: PEREIRA; SILVA, 2009, p. 5).

3.3.1 Camada *Applications*

Como o próprio nome diz, a camada *applications* ou nível 3 é onde estão as aplicações que o usuário consome, as aplicações que são desenvolvidas para usuários finais ficam todas nesta camada. O *Android* tem algumas aplicações embarcadas no sistema como um cliente de email, um programa SMS, um calendário, mapas, navegador, contatos, entre outros (AQUINO, 2007).

3.3.2 Camada *Application Framework*

Todos os desenvolvedores têm acesso completo à mesma API que é usada pelas aplicações *core* da plataforma. A arquitetura da aplicação foi projetada para simplificar a reutilização dos componentes, ou seja, qualquer componente pode disponibilizar suas capacidades e quaisquer outros componentes podem fazer uso dessas capacidades, sujeito as restrições de segurança reforçadas pelo *framework.* Esse mesmo mecanismo permite que os componentes sejam substituídos por outros componentes em tempo de desenvolvimento (AQUINO, 2007).

3.3.3 Camada *Libraries*

A arquitetura *Google Android* possui um conjunto de bibliotecas desenvolvidas em C/C++ utilizadas por diversos componentes. Suas funcionalidades são expostas através do *framework* do *Android*. A Figura 14 mostra as principais bibliotecas (AQUINO, 2007) e (FARIA, 2008).

Biblioteca	Descrição
System C Library	Uma implementação derivada da biblioteca C padrão sistema (libc) do BSD sintonizada para dispositivos rodando Linux
Media Libraries	As bibliotecas de mídia suportam os mais populares formatos de áudio, vídeo e imagem
Surface Manager	Subsistema de exibição, bem como múltiplas camadas de aplicações 2D e 3D
LibWebCore	Um *Web Browser Engine* utilizado tanto no *Android Browser* quanto para exibições *Web*
SGL	*Engine* de gráficos 2D
3D libraries	Implementação baseada no *OpenGL* As bibliotecas empregam aceleração 3D via *hardware* (quando disponível) ou o *software* de renderização 3D altamente otimizado
FreeType	Renderização de fontes *Bitmap* e *Vector*
SQLite	Um poderoso e leve *Engine* de banco de dados relacional disponível para todas as aplicações *Android*

Figura 14 – Principais bibliotecas da camada *Libraries* (In: faria, 2008)

3.3.4 Camada *Android Runtime*

Todo programa *Android* executa seu próprio processo, com sua própria instância da máquina virtual Dalvik, tal máquina foi escrita de forma que um dispositivo possa executar múltiplas máquinas virtuais concorrentemente de maneira eficaz (AQUINO, 2007).

Dalvik executa classes compiladas por um compilador da linguagem Java. Os arquivos *.class* gerados são transformados em arquivos *.dex* pela ferramenta *dx* incluída no SDK do *Android.* A máquina virtual Dalvik utiliza *kernel* de Linux para ordenar a funcionalidade de múltiplas *threads* e o gerenciamento de memória de baixo nível, assim sendo, os arquivos *.dex* são otimizados para consumo mínimo de memória (AQUINO, 2007 e FARIA, 2008).

3.3.5 Camada *Linux Kernel*

A arquitetura do *Android* é baseada no *kernel* de Linux, composta pela versão 2.6 e se responsabiliza por serviços, segurança, gerenciamento de memória e processos, rede e drivers. O *kernel* do sistema também é responsável pela abstração entre o *hardware* e o restante da pilha de softwares da plataforma (AQUINO, 2007 e FARIA, 2008).

3.4 *ANDROID SDK*

O *Android SDK* é um kit de desenvolvimento de aplicações para plataforma *Google Android,* que consiste em um emulador para simular o celular, ferramentas utilitárias e uma *API* completa para a linguagem Java, com todas as classes necessárias para desenvolver quaisquer aplicações (LECHETA, 2013).

Embora o SDK tenha um emulador que pode ser executado como um aplicativo comum, existe um *plugin* para o Eclipse que serve justamente para integrar o ambiente de desenvolvimento Java com o emulador (LECHETA, 2013).

Com tal *plugin* torna-se possível iniciar o emulador diretamente dentro do Eclipse, instalando a aplicação no emulador automaticamente e, junto ao debug do Eclipse integrado, tornando possível depurar o código fonte como qualquer outra aplicação Java (LECHETA, 2013).

Outra maneira para realizar testes nos aplicativos é plugar um celular real na porta USB do computador e executar os aplicativos diretamente no celular, sem dúvida facilita muito os testes em aparelhos reais e torna o desenvolvimento bem mais produtivo (LECHETA, 2013).

A Figura 15 ilustra o emulador *Android.*

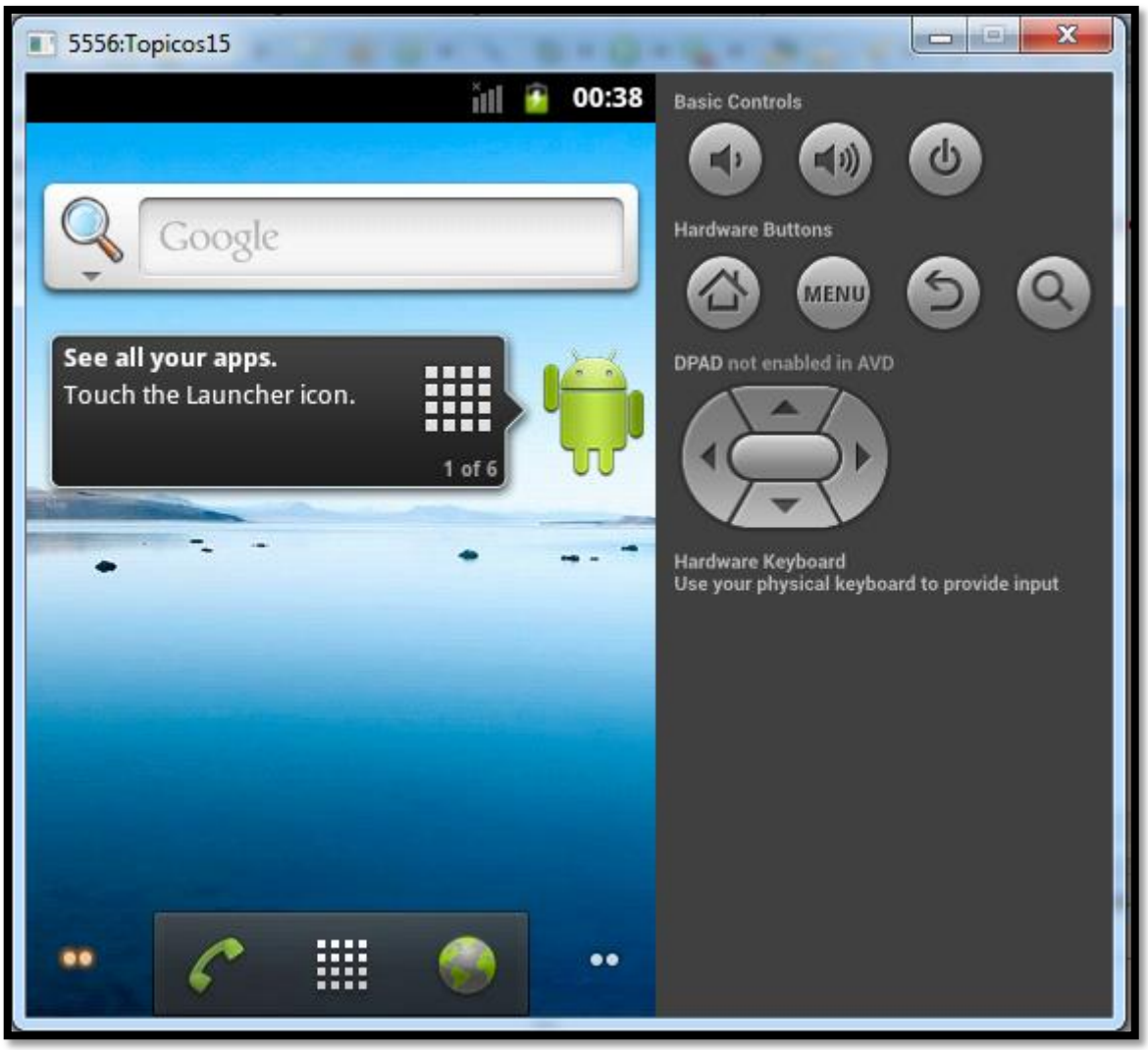

Figura 15- Emulador *Google Android*

3.5 EXEMPLOS DE APLICAÇÕES MÓVEIS

Nesta sessão serão apresentados algumas aplicações móveis e seus tipos, como aplicações corporativas e aplicações pessoais. Também serão apresentadas algumas outras plataformas do *Google Android,* tais como, plataforma *Android TV,* plataforma *Android Wear* e *Android Auto.*

3.5.1 Aplicações Móveis Corporativas

Mind Tools

Conhecimento e aprendizado nunca são demais, e tal aplicativo procura alimentar o seu cérebro, ele contém uma coletânea completa de textos sobre liderança, empreendedorismo, visão coorporativa, gerenciamento de grupos e muito mais, sempre que tiver um tempo livre é possível pegar o *smartphone* ou *tablet* e aprender mais (ENDEL11, *Mind Tools*).

Os textos são dinâmicos e fáceis de aprender, porém em inglês, contudo é necessário entender bem o idioma ou com esse aplicativos é possível ter um auxílio a mais para aprender a língua. Os textos são separados por assuntos, então é fácil encontrar o que deseja (ENDEL11, Mind Tools).

A Figura 16 ilustra uma tela do *Mind Tools*

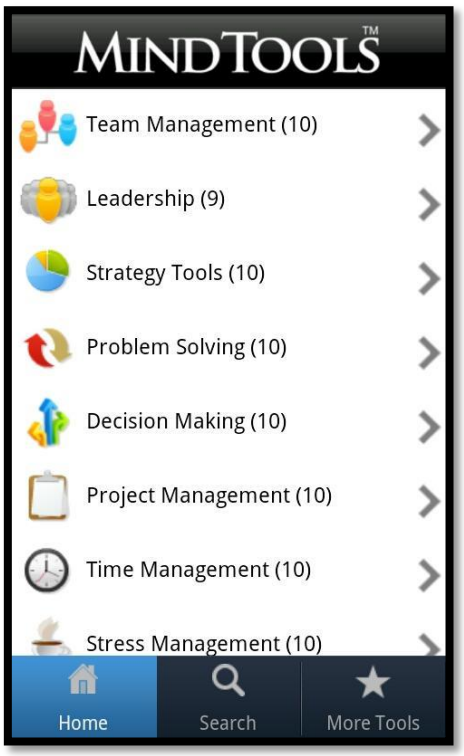

Figura 16 – Tela do *Mind Tools*

DigiCal Calendar & Widgets

Ter uma agenda digital hoje em dia é indispensável, já que os dispositivos podem ser sincronizados onde quer que esteja. Isto é, é possível acessar todos os compromissos de onde estiver, basta ter acesso a internet (ENDEL10, *DigiCal+* Calendário).

Tal aplicativo permite que a conta do *Google Calendar,* um serviço de agenda muito grande de buscar, para visualizar, editar e adicionar compromissos, mas a vantagem é que esse aplicativo possui funções extras, como *widgets* personalizáveis para a tela, que deixam os compromissos bem destacados (ENDEL10, *DigiCal+* Calendário).

A Figura 17 ilustra uma tela do *DigiCal Calendar & Widgets*

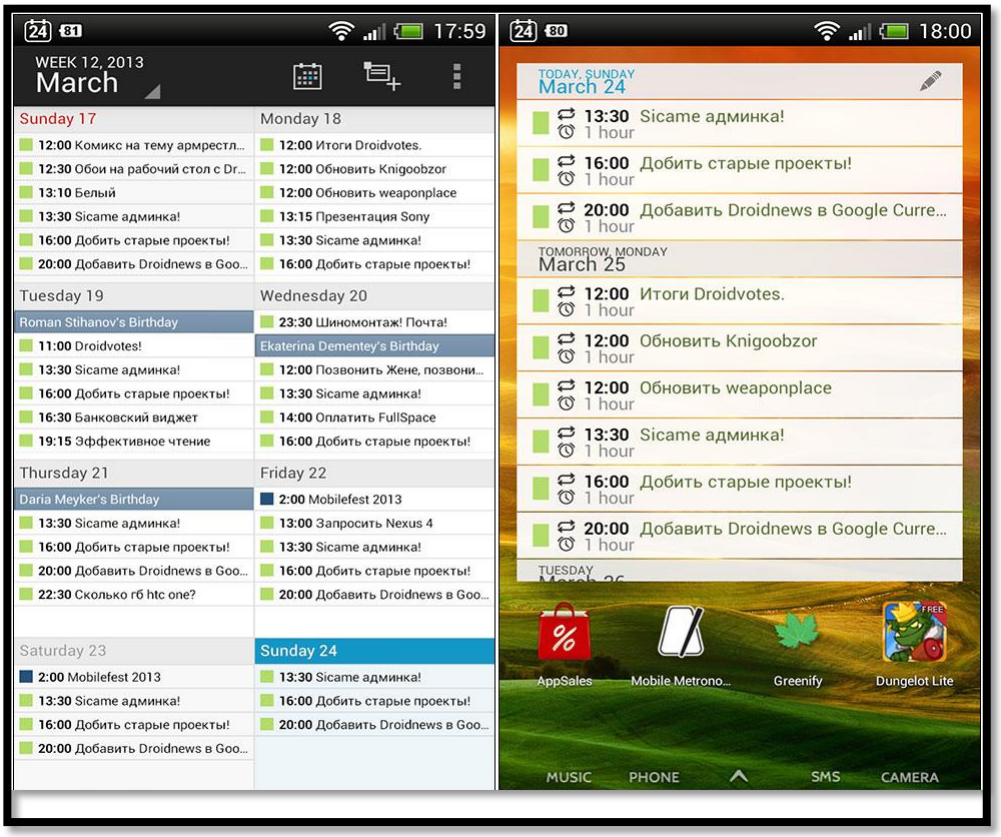

Figura 17 – Tela do *DigiCal Calendar & Widgets*

Google Drive

Este aplicativo tem uma das maneiras mais simples e pratica de guardar os seus documentos na nuvem e abrir em qualquer lugar do mundo. Para arquivos muito grandes e pesados ou mesmo para compartilhamento de itens entre os membros da equipe, o *Google Drive* é uma das melhores soluções de armazenamento que você vai encontrar para o *Android* (ENDEL12, Google Drive).

A maior vantagem do *Google Drive* é ser conectado com uma cota Google já que desta forma, é muito fácil e rápido compartilhar algo entre duas ou mais pessoas (ENDEL12, Google Drive).

A Figura 18 ilustra uma tela do *Google Drive.*

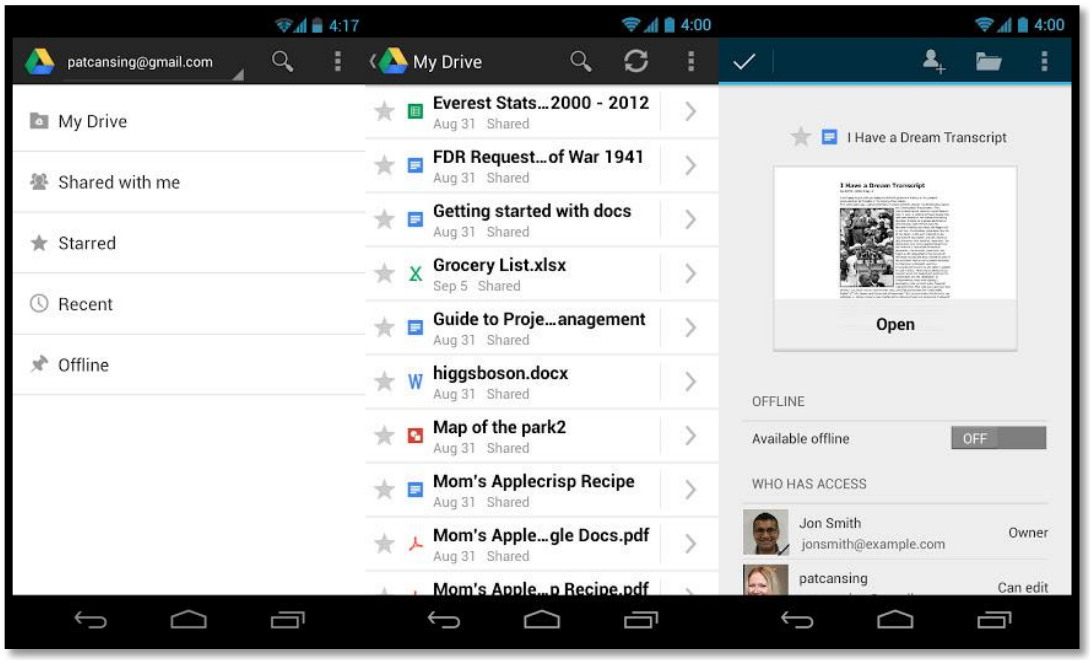

Figura 18 – Tela do *Google Drive*

3.5.2 Plataforma "*Android TV"*

É uma versão do sistema operacional *Android,* voltado especialmente para dispositivos de grandes polegadas, ou seja, voltado para TV's.Com planos para ser integrado com *SmartTV's*, consoles e *setupboxes*, o Android TV centraliza o entretenimento ao redor da conta Google do usuário (ENDEL6, 2014).

Com foco em games, a *Android* TV irá permitir executar a maioria dos jogos disponíveis na Google Play, todos serão controlados por joystick *bluetooth* ou no *Smartphone Android.* Todos os jogos poderão ser jogados no *smartphone,* na TV ou em qualquer outro dispositivo *Android,* pois os mesmos serão sincronizados e o que está sendo feito no *smartphone* terá as mesmas coisas na TV ou em qualquer outro dispositivo, também é possível jogar *multi-player on-line* ou *off-line* (ENDEL6, 2014).

3.5.3 Plataforma "*Android Wear"*

É uma versão do sistema operacional *Android*, voltado especialmente para dispositivos de pequenas polegadas, ou seja, voltado para algo que caiba no braço. O *Android Wear* organiza as informações, sugere o que você precisa e mostra tudo isso antes mesmo de você pedir, recebendo mensagens, notificações e atualizações do tempo instantaneamente (ENDEL7, 2014).

Utilizando a voz é possível responder mensagens, email, *whatsapp* e perguntar algo de nosso interesse, como "Será que vai chover neste fim de semana?", recebendo respostas diretas, tudo isso com sua voz configurada, o *Android Wear* tem a sua voz configurada (ENDEL7, 2014).

3.5.4 Plataforma *Android Auto*

Essa plataforma foi projetada tendo em mente a segurança. Com uma interface simples e intuitiva, controles integrados no volante e as novas e avançadas ações de voz, tal plataforma foi projetada para diminuir ao máximo as distrações no transito para o condutor manter o foco somente na estrada (ENDEL8, 2014).

O *Android Auto* traz automaticamente informações uteis e organizadas que aparecem apenas quando são necessárias. É fácil chegar ao destino, o Google *Maps* mostra o melhor caminho informações ao vivo sobre o tráfego, orientação sobre pista e muito mais. Seu celular pode ser conectado para ouvir as melhores músicas ou você seleciona suas músicas favoritas para viagem (ENDEL8, 2014).

4. DESENVOLVIMENTO DE COMPONENTES PARA ECLIPSE IDE

Neste capitulo serão apresentadas as principais funcionalidades e características para o desenvolvimento de componentes para Eclipse IDE, bem como suas vantagens e possibilidades de uso em projetos de desenvolvimento de software.

4.1 INTRODUÇÃO

O ambiente de desenvolvimento Eclipse IDE possibilita inúmeras modificações de interface e personalização de janelas. Dessa forma, o desenvolvedor pode contribuir por meio da concepção e implementação de novos componentes de software, conhecidos também como *plugins*, além de ser possível incorporar *plugins* de terceiros em seus projetos. O desenvolvedor pode, inclusive, oferecer pontos de extensão personalizados que podem ser prorrogados novamente porem outros desenvolvedores deixando os *plugins* conforme ele necessita (ENDEL3, 2014).

4.2 *PLUGIN DEVELOPMENT ENVIRONMENT (PDE)*

O *Plugin Development Environment* (PDE) fornece ferramentas para criar, desenvolver, testar, depurar, construir e implantar *plugins,* fragmentos, recursos, sites de atualização e produtos Eclipse RCP (ENDEL4, 2015).

PDE também fornece ferramentas OSGi abrangente, o que torna um ambiente ideal para a programação de componentes, não apenas para o desenvolvimento de *plugin* para Eclipse IDE (ENDEL4, 2015).

A maioria dos módulos de infraestrutura e serviços do Eclipse estão disponíveis como pacotes e pode ser integrado ou reutilizado. Além de que os módulos podem ser instalados e mantidos automaticamente, incluindo o *download* e atualização de

pacotes no qual alguns módulos fazem isto automaticamente, Equinox permite que o IDE possa ser atualizado sem o mesmo ser reiniciado (BALSIGER, 2010).

Muitas vezes, os *plugins* são instalados usando os chamados sites de atualização. Tal site de atualização, normalmente consiste em um *"site.xml"* que armazena as informações básicas sobre o *plugin* com sua categoria, os sistemas operacionais suportados ou o nome do recurso, o "*artifacts.xml"* oferece informações sobre cada pacote OSGi determinado por um recurso, enquanto que o *"content.xml"* contém mais informações como texto de licença e os arquivos *".jar"* necessários pelo *plugin*. O Eclipse IDE mantém muitas destas informações e arquivos *".jar"* automaticamente quando a estrutura do *plugin* é organizada de maneira correta (BALSIGER, 2010).

4.3 EXEMPLOS DE COMPONENTES EXISTENTES

Em seguida serão apresentados alguns *plugins* desenvolvidos e utilizados no Eclipse IDE.

- *Subversive* – SVN *Team Provider*: este projeto é destinado a integrar o sistema *Subversion* de controle de versão com a plataforma Eclipse. Usando este *plugin* pode-se trabalhar com projetos armazenados em repositórios *Subversion* diretamente a partir do trabalho do Eclipse de uma forma similar de trabalhar com outros repositórios de controle de versão do Eclipse, como o Git e o CVS (ENDEL5, 2013).
- *Mavem Integration for Eclipse:* M2e fornece integração abrangente *Maven* para Eclipse. O M2e pode ser utilizado para gerenciar projetos ao mesmo tempo simples e multi-módulo, também torna o desenvolvimento mais fácil através da integração dos dados de modelo de objeto de um projeto com recursos do Eclipse IDE. Com M2e, pode-se usar *Maven* no Eclipse em uma interface natural e intuitiva (ENDEL5, 2013).
- *Eclipse Color Theme:* Torna-se possível a mudança dos temas de forma conveniente e sem efeitos colaterais. Com essa ferramenta é possível mudar

a colocação da IDE para cada tipo de arquivo que será editado ou para a linguagem que será utilizada (ENDEL5, 2013).

- *Android Development Tools for Eclipse* (ADT): é um *plugin* para o Eclipse que é projetado para ter-se um poderoso ambiente no qual pode construir aplicativos *Android*. ADT amplia os recursos da IDE Eclipse para definir rapidamente novos projetos *Android*, criar uma interface de usuário, adicionar pacotes com base na API *Framework Android*, depurar aplicativos usando as ferramentas do SDK do *Android*, e até mesmo explorar arquivos assinados - ou não assinados - .apk a fim de distribuir aplicações (ENDEL5, 2013)
- *PyDev* – Python IDE for Eclipse: é utilizado para a Eclipse IDE ter suporte ao desenvolvimento em Python, também fornece recursos como depurador, console interativo, refatoração e outros (ENDEL5, 2013).
- *JBoss Tools*: é um conjunto de *plugins* para Eclipse Juno que suporta *JBoss* e tecnologias relacionadas, também tem suporte para Hibernate, JBoss AS, CDI, *Drools*, jBPM, JSF, (X)HTML, *Sam*, *Maven*, *JBoss* Portal, etc (ENDEL5, 2013).
- *Spring Tool Suite (STS) for Eclipse*: oferece o melhor ambiente de desenvolvimento Eclipse *powered* para a construção de aplicativos corporativos. STS fornece ferramentas para todas as últimas versões Java empresarial e Spring. STS apoia o direcionamento para servidores locais, virtuais e baseados em nuvem. Está disponível gratuitamente para desenvolvimento e negócios internos sem limite de tempo (ENDEL5, 2013).

5. PROPOSTA DO TRABALHO

Neste capítulo será apresentada a proposta do trabalho, que foi desenvolvido com a abordagem de *plugins* para Eclipse IDE de forma a apoiar os testes realizados nas aplicações desenvolvidas, com ênfase no teste de mutação em Java e, particularmente, em *Google Android.*

5.1 METODOLOGIA DE DESENVOLVIMENTO

No desenvolvimento da API serão utilizadas a tecnologia Java, *Google Android* e Eclipse PDE.

Java é uma linguagem de programação e uma plataforma de computação lançada pela primeira vez pela Sun Microsystems em 1995. É a tecnologia que capacita muitos programas da mais alta qualidade, como utilitários, jogos e aplicativos corporativos, entre muitos outros. O Java é executado em mais de 850 milhões de computadores pessoais e em bilhões de dispositivos em todo o mundo, inclusive telefones celulares e dispositivos de televisão. As vantagens dessa linguagem de programação são decorrentes do fato dela ser orientada a objetos, portável entre diferentes plataformas e sistemas operacionais (ENDEL13, 2015).

Google Android, como descrito no capítulo três, é uma plataforma que pode ser utilizada em vários dispositivos, como *smartphones, tablet*, *notebook*, televisão, GPS, entre outros (FARIA, 2008).

Na API que será desenvolvida, terá mutantes que fazem alterações em classes que utilizam *bundle*, altera as permissões ou *permissions,* versão mínima, máxima ou recomendada, dados móveis, utilização do cartão de memória, entre outros dentro do arquivo *AndroidManifest.xml.*

Eclipse PDE, como descrito no capítulo quatro, é uma das maneiras possíveis de estender as funcionalidades nativas do Eclipse IDE. Por meio dela, pode-se desenvolver novos *plugins* e extensões, tais como novos menus, novas janelas, novos diálogos e outros recursos.

O desenvolvimento da API será dividido em duas etapas. A primeira etapa é o desenvolvimento da API principal, na qual contém todas as regras necessárias para realizar a mutação em aplicações Java e *Google Android.* A segunda etapa é o desenvolvimento de um conjunto composto por quatro *plugins* para o ambiente do Eclipse IDE, fazendo-se uso da API de mutação desenvolvimento na primeira etapa, conforme mencionado anteriormente.

A Figura 19 ilustra a arquitetura da proposta.

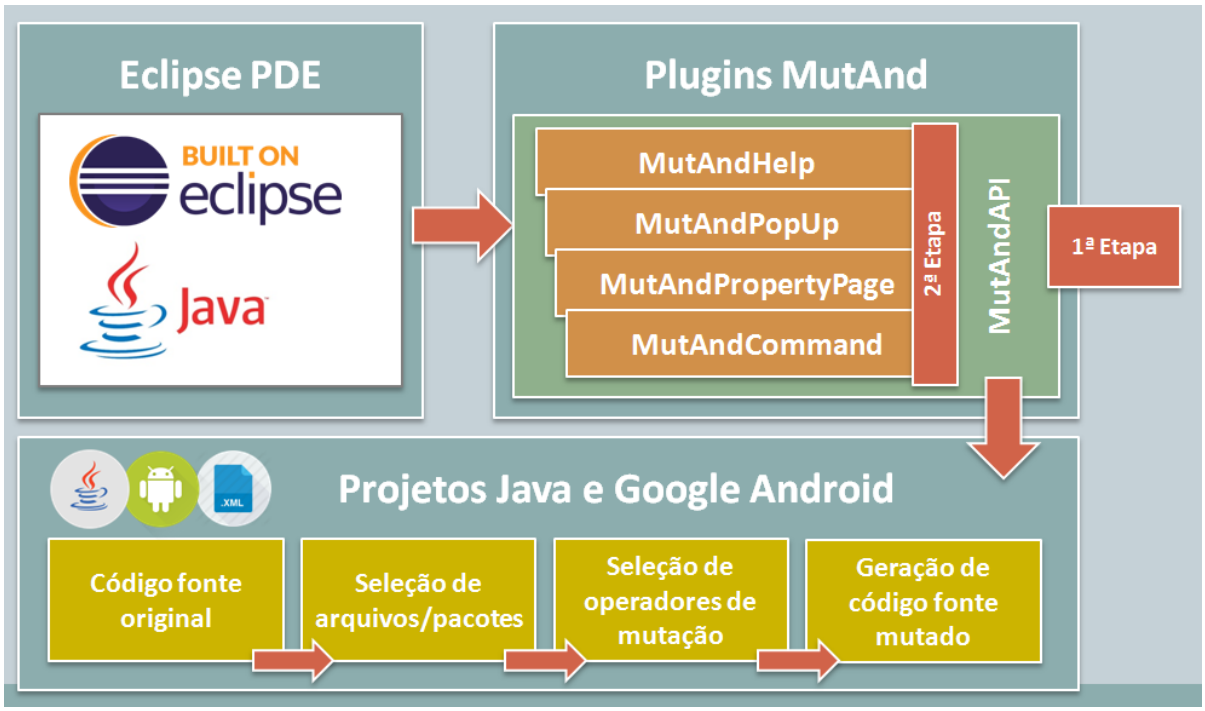

Figura 19 - Arquitetura proposta

5.2 API PROPOSTA: MutAndAPI

No desenvolvimento da proposta, inicialmente será realizada toda a parte de programação da API, onde contém todos os recursos e implementações necessários para se trabalhar com os operadores de mutação para Java e *Google Android.*

5.2.1 Operadores de mutação Java

Os operadores de mutação, no conceito de Teste de Mutação, são algoritmos que podem aplicados em programas considerados como corretos.

Há diferentes tipos e categorias de operadores de mutação, cada qual com sua mudança ou mutação específica. Entre algumas das categorias, há operadores de mutação aritméticos, condicionais, incrementais, entre outros, que podem ser aplicados em diferentes programas de diferentes plataformas e linguagens de programação.

Entretanto, também é possível definir e implementar operadores de mutação específicos para plataformas e linguagens de programação. Por exemplo, pode-se propor um conjunto de operadores de mutação que são empregados especificamente para programas Java e/ou *Google Android*.

Os operadores de mutação Java implementados foram os mesmos apresentados na seção 2.3.2 Operadores de Mutação e Exemplos de Uso.

5.2.2 Operadores de mutação *Google Android*

Nesta seção será apresentado os operadores de mutação *Google Android* que será desenvolvido.

5.2.2.1 Activity Declaration Deletion

O operador de mutação *Activity Declaration Deletion* realiza a remoção de entradas de uma *Activity* específica no arquivo AndroidManifest.xml. A Figura 20 exemplifica o código fonte original e a Figura 21 exemplifica o código fonte modificado conforme a descrição.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.edu.fema.sistemaacademico"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="10"
      android:targetSdkVersion="10" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity android:name=".view.PrincipalActivity" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".view.CursosListActivity" >
      </activity>
      <activity android:name=".view.CursoActivity" >
      </activity>
      <activity android:name=".view.AlunosListActivity" >
      </activity>
      <activity android:name=".view.AlunoActivity" >
      </activity>
   </application>

</manifest>
```

Figura 20 – Código fonte original AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.edu.fema.sistemaacademico"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="10"
      android:targetSdkVersion="10" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity android:name=".view.PrincipalActivity" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
```

```
            <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
      </activity>
      <activity android:name=".view.CursosListActivity" >
      </activity>
      <activity android:name=".view.CursoActivity" >
      </activity>
      <activity android:name=".view.AlunosListActivity" >
      </activity>

   </application>

</manifest>
```

Figura 21 – Código fonte modificado após a mutação *Activity Declaration Deletion*

5.2.2.2 Bundle Object Deletion

O operador de mutação *Bundle Object Deletion* realiza a remoção de entradas de valores, ou seja, parâmetros em objetos de referência do tipo *Bundle.* A Figura 22 exemplifica o código fonte original e a Figura 23 exemplifica o código fonte modificado conforme a descrição.

```
Bundle parametros = new Bundle();
parametros.putSerializable("newname", aluno);

Bundle parametros = new Bundle();
parametros.putTeste("newname", aluno);

Bundle parametros = new Bundle();
parametros.putQualquer("newname", aluno);

Bundle parametros = new Bundle();
parametros.putExtra("newname", aluno);

Bundle parametros = new Bundle();
parametros.putBcc("newname", aluno);
```

Figura 22 – Código fonte original

```
Bundle parametros = new Bundle();
```

```
Bundle parametros = new Bundle();


Bundle parametros = new Bundle();


Bundle parametros = new Bundle();


Bundle parametros = new Bundle();
```

Figura 23 – Código fonte modificado após a mutação *Bundle Object Deletion*

5.2.2.3 Bundle Object Key Modification

O operador de mutação *Bundle Object Key Modification* realiza a alteração na chave de objetos de referência do tipo *Bundle.* Dessa forma, parâmetros ainda são passados, porém com chaves de valores diferentes da versão original. A Figura 24 exemplifica o código fonte original e a Figura 25 exemplifica o código fonte modificado conforme a descrição.

```
Bundle parametros = new Bundle();
parametros.putSerializable("newname", aluno);

Bundle parametros = new Bundle();
parametros.putTeste("newname", aluno);

Bundle parametros = new Bundle();
parametros.putQualquer("newname", aluno);

Bundle parametros = new Bundle();
parametros.putExtra("newname", aluno);

Bundle parametros = new Bundle();
parametros.putBcc("newname", aluno);
```

Figura 24 – Código fonte original

```
Bundle parametros = new Bundle();
parametros.putSerializable("", aluno);
```

```
Bundle parametros = new Bundle();
parametros.putTeste("", aluno);

Bundle parametros = new Bundle();
parametros.putQualquer("", aluno);

Bundle parametros = new Bundle();
parametros.putExtra("", aluno);

Bundle parametros = new Bundle();
parametros.putBcc("", aluno);
```

Figura 25 – Código fonte modificado após a mutação *Bundle Object Key Modification*

5.2.2.4 Database Operation Exception

O operador de mutação *Database Operation Exception* realiza a inclusão de instruções que lançam exceções (*Java Exceptions*) antes de uma operação de banco de dados SQLite ser realizada. Dessa forma, as instruções de inserção *("db.insert*"), alteração ("*db.update*"), remoção ("*db.delete*") e seleção ("*db.query*") não serão invocadas corretamente. A Figura 26 exemplifica o código fonte original e a Figura 27 exemplifica o código fonte modificado conforme a descrição.

```
db.insert(IAluno.TABLE_NAME, null, values);

db.update(IAluno.TABLE_NAME, values, IAluno.COLUMN_RA + "=?", new String[] {
Long.toString(aluno.getRa()) });

db.delete(IAluno.TABLE_NAME, IAluno.COLUMN_RA + "=?", new String[] { Long.toString(ra) });

Cursor cursor = db.query(IAluno.TABLE_NAME, new String[] { IAluno.COLUMN_NOME,
IAluno.COLUMN_DT_NASC, IAluno.COLUMN_SEXO, IAluno.COLUMN_CURSO,
IAluno.COLUMN_FG_ATIVO }, IAluno.COLUMN_RA + "=?", new String[] { Long.toString(ra) }, null,
null, null);
```

Figura 26 – Código fonte original

```
if (true) {throw new IllegalArgumentException("generated error");}
db.insert(IAluno.TABLE_NAME, null, values);
```

```
if (true) {throw new IllegalArgumentException("generated error");}
db.update(IAluno.TABLE_NAME, values, IAluno.COLUMN_RA + "=?", new String[] {
Long.toString(aluno.getRa()) });

if (true) {throw new IllegalArgumentException("generated error");}
db.delete(IAluno.TABLE_NAME, IAluno.COLUMN_RA + "=?", new String[] { Long.toString(ra) });

if (true) {throw new IllegalArgumentException("generated error");}
Cursor cursor = db.query(IAluno.TABLE_NAME, new String[] { IAluno.COLUMN_NOME,
IAluno.COLUMN_DT_NASC, IAluno.COLUMN_SEXO, IAluno.COLUMN_CURSO,
IAluno.COLUMN_FG_ATIVO }, IAluno.COLUMN_RA + "=?", new String[] { Long.toString(ra) }, null,
null, null);
```

Figura 27 – Código fonte modificado após a mutação *Database Operation Exception*

5.2.2.5 Database Type Modification

O operador de mutação *Database Type Modification* realiza a mudança no tipo de instância do banco de dados. Os tipos que podem ser alternados são *readableDatabase* e *writableDatabase*. Respectivamente, os valores definem que o banco de dados é somente leitura e o banco de dados é leitura/gravação. A Figura 28 exemplifica o código fonte original e a Figura 29 exemplifica o código fonte modificado conforme a descrição.

```
package br.edu.fema.sistemaacademico.db;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import br.edu.fema.sistemaacademico.constants.IAluno;
import br.edu.fema.sistemaacademico.constants.ICurso;

public class DatabaseHelper {

    private DBHelper helper = null;
    private SQLiteDatabase db = null;

    public DatabaseHelper(Context context) {
        helper = new DBHelper(context);
    }

    public SQLiteDatabase open() {
        return db = helper.getWritableDatabase();
```

```
        //o de baixo é só para deixar para teste
        //return db = helper.getReadableDatabase();
    }

    public void close() {
        db.close();
    }

    public class DBHelper extends SQLiteOpenHelper {

        public DBHelper(Context context) {
            super(context, DatabaseConstants.DB_NAME, null, DatabaseConstants.VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            onUpgrade(db, 0, DatabaseConstants.VERSION);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            if (oldVersion + 1 > newVersion) {
                return;
            } else {
                switch (oldVersion + 1) {
                    case 1:
                        db.execSQL(ICurso.CREATE_TABLE);
                        break;
                    case 2:
                        db.execSQL(IAluno.CREATE_TABLE);
                        break;
                }

                onUpgrade(db, ++oldVersion, newVersion);
            }
        }

    }

}
```

Figura 28 – Código fonte original

```
package br.edu.fema.sistemaacademico.db;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import br.edu.fema.sistemaacademico.constants.IAluno;
import br.edu.fema.sistemaacademico.constants.ICurso;

public class DatabaseHelper {

    private DBHelper helper = null;
    private SQLiteDatabase db = null;
```

```
    public DatabaseHelper(Context context) {
        helper = new DBHelper(context);
    }

    public SQLiteDatabase open() {
        return db = helper.getReadableDatabase();
        //o de baixo Ã© sÃ³ para deixar para teste
        //return db = helper.getWritableDatabase();
    }

    public void close() {
        db.close();
    }

    public class DBHelper extends SQLiteOpenHelper {

        public DBHelper(Context context) {
            super(context, DatabaseConstants.DB_NAME, null, DatabaseConstants.VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            onUpgrade(db, 0, DatabaseConstants.VERSION);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            if (oldVersion + 1 > newVersion) {
                return;
            } else {
                switch (oldVersion + 1) {
                    case 1:
                        db.execSQL(ICurso.CREATE_TABLE);
                        break;
                    case 2:
                        db.execSQL(IAluno.CREATE_TABLE);
                        break;
                }

                onUpgrade(db, ++oldVersion, newVersion);
            }
        }

    }

}
```

Figura 29 – Código fonte modificado após a mutação *Database Type Modification*

5.2.2.6 Permission Declaration Deletion

O operador de mutação *Permission Declaration Deletion* realiza a remoção de entradas de permissões de uso da aplicação *Google Android* no arquivo AndroidManifest.xml. A Figura 30 exemplifica o código fonte original e a Figura 31 exemplifica o código fonte modificado conforme a descrição.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.edu.fema.sistemaacademico"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="10"
      android:targetSdkVersion="10" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity android:name=".view.PrincipalActivity" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".view.CursosListActivity" >
      </activity>
      <activity android:name=".view.CursoActivity" >
      </activity>
      <activity android:name=".view.AlunosListActivity" >
      </activity>
      <activity android:name=".view.AlunoActivity" >
      </activity>
   </application>

   <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

</manifest>
```

Figura 30 - Código fonte original

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.edu.fema.sistemaacademico"
   android:versionCode="1"
   android:versionName="1.0" >
```

```
   <uses-sdk
      android:minSdkVersion="10"
      android:targetSdkVersion="10" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity android:name=".view.PrincipalActivity" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".view.CursosListActivity" >
      </activity>
      <activity android:name=".view.CursoActivity" >
      </activity>
      <activity android:name=".view.AlunosListActivity" >
      </activity>
      <activity android:name=".view.AlunoActivity" >
      </activity>
   </application>

   <uses-permission android:name="android.permission.INTERNET" />

</manifest>
```

Figura 31 - Código fonte modificado após a mutação *Permission Declaration Deletion*

5.2.2.7 Permission Declaration Insertion

O operador de mutação *Permission Declaration Insertion* realiza a inclusão de entradas de permissões de uso da aplicação *Google Android* no arquivo AndroidManifest.xml. A Figura 32 exemplifica o código fonte original e a Figura 33 exemplifica o código fonte modificado conforme a descrição.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.edu.fema.sistemaacademico"
   android:versionCode="1"
   android:versionName="1.0" >
```

```
    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="10" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity android:name=".view.PrincipalActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".view.CursosListActivity" >
        </activity>
        <activity android:name=".view.CursoActivity" >
        </activity>
        <activity android:name=".view.AlunosListActivity" >
        </activity>
        <activity android:name=".view.AlunoActivity"></activity>
    </application>

</manifest>
```

Figura 32 - Código fonte original

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.edu.fema.sistemaacademico"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="10" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity android:name=".view.PrincipalActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".view.CursosListActivity" >
        </activity>
        <activity android:name=".view.CursoActivity" >
        </activity>
        <activity android:name=".view.AlunosListActivity" >
        </activity>
```

```
      <activity android:name=".view.AlunoActivity"></activity>
   </application>

   <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Figura 33 - Código fonte modificado após a mutação *Permission Declaration Insertion*

5.2.2.8 Version Modification

O operador de mutação *Version Modification* realiza a mudança nos atributos de versões da aplicação *Google Android* no arquivo AndroidManifest.xml. Os atributos que podem ser modificados são *android*:*minSdkVersion* (mínima versão recomendável), *android*:*maxSdkVersion* (máxima versão recomendável) e *android*:*targetSdkVersion* (versão de compilação recomendável). A Figura 34 exemplifica o código fonte original e a Figura 35 exemplifica o código fonte modificado conforme a descrição.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.edu.fema.sistemaacademico"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="10"
      android:maxSdkVersion="20"
      android:targetSdkVersion="10" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity android:name=".view.PrincipalActivity" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".view.CursosListActivity" > </activity>
      <activity android:name=".view.CursoActivity" >
      </activity>
```

```
      <activity android:name=".view.AlunosListActivity" >
      </activity>
      <activity android:name=".view.AlunoActivity" >
      </activity>
   </application>
   <uses-permission android:name="android.permission.INTERNET" />

</manifest>
```

Figura 34 - Código fonte original

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.edu.fema.sistemaacademico"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="7"
      android:maxSdkVersion="20"
      android:targetSdkVersion="10" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity android:name=".view.PrincipalActivity" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".view.CursosListActivity" > </activity>
      <activity android:name=".view.CursoActivity" >
      </activity>
      <activity android:name=".view.AlunosListActivity" >
      </activity>
      <activity android:name=".view.AlunoActivity" >
      </activity>
   </application>
   <uses-permission android:name="android.permission.INTERNET" />

</manifest>
```

Figura 35 - Código fonte modificado após a mutação *Version Modification*

5.2.2.9 View Instance Nullable

O operador de mutação *View Instance Nullable* realiza a mudança em todas as instruções que fazem uso do método *findViewById* para obter uma instância de um dado componente visual. Dessa forma, todas as instâncias de classes *Activities* serão nulas (valor de "*null*") e não será possível invocar manipular os componentes na tela. A Figura 36 exemplifica o código fonte original e a Figura 37 exemplifica o código fonte modificado conforme a descrição.

```
@Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.aluno_view);

      txtRa = (EditText) findViewById(R.id.txtRa);
      txtNome = (EditText) findViewById(R.id.txtNome);
      txtDataNascimento = (EditText) findViewById(R.id.txtDataNascimento);
      btnSelecionarData = (ImageButton) findViewById(R.id.btnSelecionarData);
      rdgSexo = (RadioGroup) findViewById(R.id.rdgSexo);
      txtDescricaoCurso = (EditText) findViewById(R.id.txtDescricaoCurso);
      btnPesquisarCurso = (ImageButton) findViewById(R.id.btnPesquisarCurso);
      ckAtivo = (CheckBox) findViewById(R.id.ckAtivo);
```

Figura 36 - Código fonte original

```
@Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.aluno_view);

      txtRa = null;
      txtNome = null;
      txtDataNascimento = null;
      btnSelecionarData = null;
      rdgSexo = null;
      txtDescricaoCurso = null;
      btnPesquisarCurso = null;
      ckAtivo = null;
```

Figura 37 - Código fonte modificado após a mutação *View Instance Nullable*

5.3 OBJETIVO DOS *PLUGINS*

Espera-se que, ao final, com os *plugins* desenvolvidos, os mesmos sejam capazes de executar todos os operadores de mutação desenvolvido, permitindo a seleção de quais operadores de mutação Java pretende-se executar em determinada classe, permitir a seleção de quais operadores de mutação *Google Android* pretende-se executar para determinado arquivo *Google Android*, no caso de operadores *Google Android* permitir a definição dos valores extras que necessitam para mutação e por fim, permitir a seleção do diretório de saída de tais arquivos.

5.4 *PLUGINS* PROPOSTOS

Serão desenvolvidos quatro *plugins,* sendo eles MutAndCommand, MutAndHelp, MutAndPopUp e MutAndPropertyPage. Nas próximas subseções será explicado cada um deles.

5.4.1 MutAndCommand

O MutAndCommand é um *plugin* que acrescenta um botão na barra de ferramentas do Eclipse IDE, tal botão pode fazer a função que a pessoa que o criou quer, nesse caso em especifico foi criado um botão, que por sua vez ao ser acionado abre uma outra tela com informações do MutAnd.

5.4.2 MutAndHelp

O MutAndHelp acrescenta um book de help no catálogo de ajuda do Eclipse IDE, ele pode ser acessado pelo menu Help e logo em seguida escolher a opção Help *Contents* ou no menu MutAnd e logo em seguida escolher a opção ajuda. Ao abrir a

tela de Help, é possível escolher a opção MutAnd – *Plugins* de apoio do teste de mutação onde está descrito toda a ajuda necessária para os usuários do *plugin,* como o que são mutantes, o que é teste de mutação, o que são operadores de mutação, como utilizar o teste de mutação, o antes e o depois do código sofrer a mutação, entre outros.

5.4.3 MutAndPopUp

O MutAndPopUp acrescenta uma opção no menu de contexto do Eclipse IDE, ou seja, quando clicar com o botão direito sobre um arquivo XML, um arquivo Java ou até mesmo sobre um pacote Java, onde considera todos os arquivos do pacote, abre uma janela para selecionar quais operadores de mutação serão executados. Esse é o *plugin* principal da API, pois com ele é possível escolher quais operadores de mutação Java e/ou *Google Android* serão executados, no caso de *Google Android* é permitido colocar os extras em alguns casos. Logo em seguida é possível selecionar qual será a pasta na qual os arquivos modificados serão salvos e por fim é realizado a execução para ser gerado os mutantes.

5.4.4 MutAndPropertyPage

O MutAndPropertyPage acrescenta uma página customizada ao abrir as propriedades de um arquivo, ao clicar com o botão direito em um arquivo Java ou XML e selecionar *Properties*, na janela que abre é possível ver uma opção MutAnd PropertyPage onde fica salvo o histórico de execução dos operadores de mutação realizado naquele arquivo, Dessa forma é possível fazer um controle e uma análise dos operadores executados em cada arquivo, para que no final dos testes, ser possível ter certeza que os testes que foram planejados fazer em cada arquivo realmente foram feitos.

5.5 CONSIDERAÇÕES FINAIS E LIMITAÇÕES DA PROPOSTA

O objetivo da proposta deste trabalho é de propor e desenvolver uma API com um conjunto de *plugins* para Eclipse IDE, baseado na plataforma Eclipse PDE, que apoie o uso de operadores de mutação em projetos Java e *Google Android*. Junto ao projeto, foram propostos alguns operadores de mutação em *Google Android*, ainda não existentes. Entretanto, as classes e arquivos resultantes da mutação serão gerados em um diretório de saída, a escolha do usuário.

Em trabalhos futuros, espera-se que os códigos fontes modificados sejam reincorporados ao projeto de origem e, dessa forma, seja possível executar a aplicação com o código fonte gerado pelos operadores de mutação. Assim, a limitação da proposta da pesquisa se baseia na ideia de que o escopo é a proposta, desenvolvimento e execução dos operadores de mutação, dos *plugins* e das ferramentas geradas. Nesse momento, não se espera validar e executar os arquivos gerados.

6. ESTUDO DE CASO

Neste capitulo serão apresentados as especificações e implementações da proposta do trabalho descrita no capítulo anterior. Este capítulo irá explicar e demonstrar a API desenvolvida MutAndAPI e suas funcionalidades, bem como todos os *plugins* e artefatos desenvolvidos com Eclipse PDE.

6.1 MutAndAPI: API DE APOIO AO TESTE DE MUTAÇÃO PARA APLICAÇÕES MÓVEIS

A API como descrito anteriormente foi dividido em duas etapas, sendo que na primeira etapa foi a implementação da API, onde tem todas as suas funcionalidades para o teste de mutação e operadores de mutação para Java e *Google Android.* A estrutura do projeto de tal etapa foi dividido em pacotes, sendo eles o pacote de *operators* onde tem todos os operadores de mutação Java descrito no capítulo dois, o pacote *operators.android* onde tem todos os operadores de mutação *Google Android* descrito no capítulo cinco, o pacote *util* onde tem classes para manipulação de arquivos, ou seja, leitura e escrita de arquivos textos, esses arquivos podem ser .java, .xml, .txt, entre outros e o pacote *main* onde tem a classe principal que por sua vez chama todas as outras classes para a realização da mutação.

A Figura 38 ilustra o código fonte da classe principal, que por sua vez instancia uma variável do tipo *file* cujo nome é arquivoOriginal atribuindo a mesma um arquivo, no caso DatabaseOperationException.txt que está no caminho \\MutantAPI_ver4\\src-examples\\ DatabaseOperationException.txt, logo em seguida instancia outra variável do tipo *file* chamada arquivoDestino e passa seu caminho, caso o arquivo não exista o mesmo é criado automaticamente, depois, é instanciado uma variável do tipo *String* chamada conteudoOriginal que por sua vez recebe a leitura do arquivo chamado arquivoOriginal, tal leitura é feita pelo método *readFile* que está dentro da classe FileUtil, em seguida é instanciado uma *String* chamada conteudoMutado que invoca o método mutator da classe Mutator passando como parâmetro o conteúdo

que será modificado e qual operador de mutação será utilizado, é nesse momento que a mutação é realizada.

```
File arquivoOriginal = new File("..\\MutantAPI_ver4\\src-
examples\\DatabaseOperationException.txt");
File arquivoDestino = new File("..\\MutantAPI_ver4\\source-
output\\DatabaseOperationException.txt");

String conteudoOriginal = FileUtil.readFile(arquivoOriginal);

String      conteudoMutado      =      new      Mutator().mutate(conteudoOriginal,
Operator.DATA_BASE_OPERATION_EXCEPTION);
```

Figura 38 – Parte do código fonte da classe MainMutant

A Figura 39 se refere a parte de um método dentro do pacote *operators.android,* tal método instancia uma *Stirng* com o nome conteudoMutado passando para a mesma o conteúdo original do arquivo que vem com argumento nesse método, o conteudoMutado substitui tudo o que passa por tal expressão regular por [a] e logo em seguida troca [a] por (vazio), nesse caso remove a declaração de *activitys*.

```
String conteudoMutado = new String(conteudoOriginal);

conteudoMutado = conteudoMutado.replaceAll
("<\\s*activity\\s*android:name\\s*=\\s*"+extra+"\\s*>[ ]*\\s*<[ ]*/[ ]*activity[
]*>[ ]*", "[a]");

conteudoMutado = conteudoMutado.replaceAll("\\[a\\]", " ");
```

Figura 39 – Parte do código fonte da classe ActivityDeclarationDeletionMutator

6.2 MutAndCommand: *PLUGIN* DE MENU PARA O MutAndAPI

MutAndCommand foi um *plugin* desenvolvido onde mostras algumas informações básicas sobre a MutAndAPI como o que é, quem desenvolveu, contatos das pessoas que desenvolveram, nome da instituição, site da instituição, entre outras.

A Figura 40 mostra um pouco do código fonte para realizar o desenvolvimento do MutAndCommand. Este código fonte contém uma *StirngBuilder* cujo nome é informacoes e é necessário preencher ela com quais informações irão apareçer na tela utilizando seu método *append*. Depois é realizado a chamada de uma tela, atribui um nome à ela e quais informações serão exibidas nela, nesse caso as informações são tudo o que tem dentro da *StringBuilder informacoes.*

```
public Object execute(ExecutionEvent event) throws ExecutionException {
	StringBuilder informacoes = new StringBuilder("");

	informacoes.append("MutAnd - Plugins de apoio ao teste de mutação em
aplicações móveis");
	informacoes.append("\n\n");
	informacoes.append("Joel Rodrigues Alvares Leal
(joel_19lakers@hotmail.com)");
	informacoes.append("\n");
	informacoes.append("Prof. Guilherme de Cleva Farto
(guilherme.farto@gmail.com)");
	informacoes.append("\n\n");
	informacoes.append("Fundação Educacional do Município de Assis (FEMA)");
	informacoes.append("\n");
	informacoes.append("http://www.fema.edu.br");
	informacoes.append("\n\n");
	informacoes.append("Instituto Municipal de Ensino Superior de Assis
(IMESA)");
	informacoes.append("\n\n");
	informacoes.append("Assis, São Paulo, Brasil - 2015");

	IWorkbenchWindow window=HandlerUtil.getActiveWorkbenchWindowChecked(event);

	MessageDialog.setDefaultImage(Activator.getDefault().getImageRegistry().get(
Activator.ICON_B));

	MessageDialog.openInformation(window.getShell(), "MutAnd - Informações",
informacoes.toString());

		return null;
	}
```

Figura 40 – Método *execute* da classe MutAndHandler

6.3 MutAndHelp: *PLUGIN* DE AJUDA PARA O MutAndAPI

MutAndHelp foi um *plugin* desenvolvido para auxiliar os usuários do MutAndAPI, tal *plugin* tem todas as ajudas necessárias que o usuário necessita para a utilização do

plugin. Para sua utilização é necessário acessar o *Help Contents* dentro do menu *Help* do Eclipse ou acessar através do menu *MutAnd,* sub menu *ajuda.*

Dentro dessa ajuda tem todas as informações necessárias para saber o que é teste de mutação, mutante, operador de mutação, quais são os operadores existentes, como utilizar cada operador de mutação, entre outras informações.

A Figura 41 mostra uma parte do código fonte escrito para o desenvolvimento de tal *plugin*, onde é possível inserir o título e quais os tópicos e sub tópicos existem dentro de tal ajuda.

```
<toc label="MutAnd - Plugins de apoio ao teste de mutação" topic="html/toc.html">
   <topic label="Teste de Mutação">
      <anchor id="testedemutacao"/>
   </topic>
   <topic label="Operadores de Mutação">
      <anchor id="operadoresdemutacao"/>
   </topic>
   <topic label="Mutantes">
      <anchor id="mutantes"/>
   </topic>
   <topic label="Operadores de Mutação do MutAnd">
      <anchor id="operadoresdemutacaomutand"/>
   </topic>
</toc>
```

Figura 41 – toc.xml dentro do MutAndHelp

A Figura 42 mostra o código HTML feito para a introdução do tópico de operadores de mutação dentro da MutAndHelp.

```
<!doctype html>
<html>
<head>
<title>HTML Editor - Full Version</title>
</head>
<body>
       <h2>
              <font face="verdana, geneva, sans-serif">Operadores de
                     Muta&ccedil;&atilde;o</font>
       </h2>
```

```
	<h3>
		<span style="font-family: verdana, geneva, sans-serif;">A
			import&acirc;ncia dos operadores de
muta&ccedil;&atilde;o</span>
	</h3>

	<hr />

	<p>
		<font face="verdana, geneva, sans-serif"><span
			style="font-size: 12px;">Esta se&ccedil;&atilde;o descreve os
				conceitos dos operadores de muta&ccedil;&atilde;o,
principal
				elemento do Teste  de
Muta&ccedil;&atilde;o.</span></font>
	</p>
</body>
</html>
```

Figura 42 – Introdução dentro do tópico de operadores de mutação

6.4 MutAndPopUp: *PLUGIN* DE SELEÇÃO E EXECUÇÃO DE OPERADORES DE MUTAÇÃO PARA O MutAndAPI

Como descrito anteriormente, o MutAndPopUp é o *plugin* principal pois com ele que é possível realizar o teste de mutação fazendo as modificações necessárias para cada arquivo, tal *plugin* acrescenta um item no menu de contexto do Eclipse IDE para realizar o teste de mutação em uma classe, arquivo XML ou em um pacote completo.

A Figura 43 mostra o código fonte para criar o menu de contexto do Eclipse IDE, atribuindo ao nome do menu MutAnd e ao sub menu Operadores de Mutação, que por sua vez quando selecionado invoca um objeto da classe MutAndMainAction que está no pacote *mutand.popup.actions.*

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
	<extension point="org.eclipse.ui.popupMenus">
		<objectContribution
	  id="MutAndPopUp.contribution1"
```

```
         objectClass="org.eclipse.core.resources.IFile">
                    <menu
            icon="images/icon_a.gif"
            id="MutAndPopUp.menu1"
            label="MutAnd"
            path="additions">
                          <separator
               name="group1">
                          </separator>
                    </menu>
                    <action
            class="mutand.popup.actions.MutAndMainAction"
            enablesFor="1"
            icon="images/icon_b.gif"
            id="MutAndPopUp.newAction"
            label="Operadores de Mutação"
            menubarPath="MutAndPopUp.menu1/group1">
                    </action>
              </objectContribution>
              <objectContribution
         id="MutAndPopUp.contribution2"
         objectClass="org.eclipse.jdt.core.IJavaElement">
                    <menu
            icon="images/icon_a.gif"
            id="MutAndPopUp.menu2"
            label="MutAnd"
            path="additions">
                          <separator
               name="group2">
                          </separator>
                    </menu>
                    <action
            class="mutand.popup.actions.MutAndMainAction"
            enablesFor="1"
            icon="images/icon_b.gif"
            id="MutAndPopUp.newAction"
            label="Operadores de Mutação"
            menubarPath="MutAndPopUp.menu2/group2">
                    </action>
              </objectContribution>
       </extension>
</plugin>
```

Figura 43 – *plugin*.xml do projeto MutAndPopUp

A Figura 44 mostra o código fonte de como é salvo o histórico de mutações para ser acessado pelo MutAndPropertyPage, tal histórico é salvo como metadado do arquivo realizado a mutação, esse metadado não é visível, porém existe um método que busca ele para ser mostrado, tal método será apresentado na próxima seção.

```
for (IResource resource : resources) {
	listaHistoricoDeMutacoes.clear();

	listaHistoricoDeMutacoes.add(historicoDeMutacaoVO);

	String historicoDeMutacoes = resource.getPersistentProperty(new
QualifiedName("", HISTORICO_MUTACOES));

	if(historicoDeMutacoes != null && !"".equalsIgnoreCase(historicoDeMutacoes))
{
	List<HistoricoDeMutacaoVO> listaPivot =
mutand.core.GsonBuilder.getGson().fromJson(historicoDeMutacoes, new
TypeToken<List<HistoricoDeMutacaoVO>>() {}.getType());

	listaHistoricoDeMutacoes.addAll(listaPivot);
}
	resource.setPersistentProperty(new QualifiedName("", HISTORICO_MUTACOES),
		gson.toJson(listaHistoricoDeMutacoes));

}
```

Figura 44 – Código fonte que salva o histórico de mutação no metadado do arquivo

6.5 MutAndPropertyPage: *PLUGIN* DE HISTÓRICO DE OPERADORES DE MUTAÇÃO PARA O MutAndAPI

MutAndPropertyPage foi desenvolvido para que a pessoa que estiver realizando o teste em alguma aplicação tenha registrado em algum local quais foram os testes realizados tais classes, pacotes, entre outros. Tal *plugin* adiciona a opção MutAndPropertyPage dentro das propriedades de um arquivo XML, Java ou qualquer outro arquivo que foi realizado o teste de mutação. Sempre que qualquer teste de mutação é realizado existe uma classe que salva os testes, como descrito na seção anterior. A Figura 45 mostra o código fonte onde é realizado a busca dos metadados da classe.

```
String owner = getSelectedResource().getPersistentProperty(new QualifiedName("",
HISTORICO_MUTACOES));
```

```
                txtHistoricoDeMutacoes.setText((owner != null) ? owner : "");
```

Figura 45 – Código fonte que busca dentro do metadado do arquivo o histórico do teste de mutação

6.6 Exemplificação do uso do MutAndAPI e *plugins* desenvolvidos

Para utilização do MutAndAPI é necessário ter o jar MutantAPI acoplado no projeto que necessita do *plugin.*

Para acessar as informações do *plugin* é necessário ir no menu MutAnd e selecionar a opção informações ou na barra de ferramenta selecionar a opção MutAnd-Informações. A Figura 46 mostra as duas maneiras de verificar as informações do *plugin.*

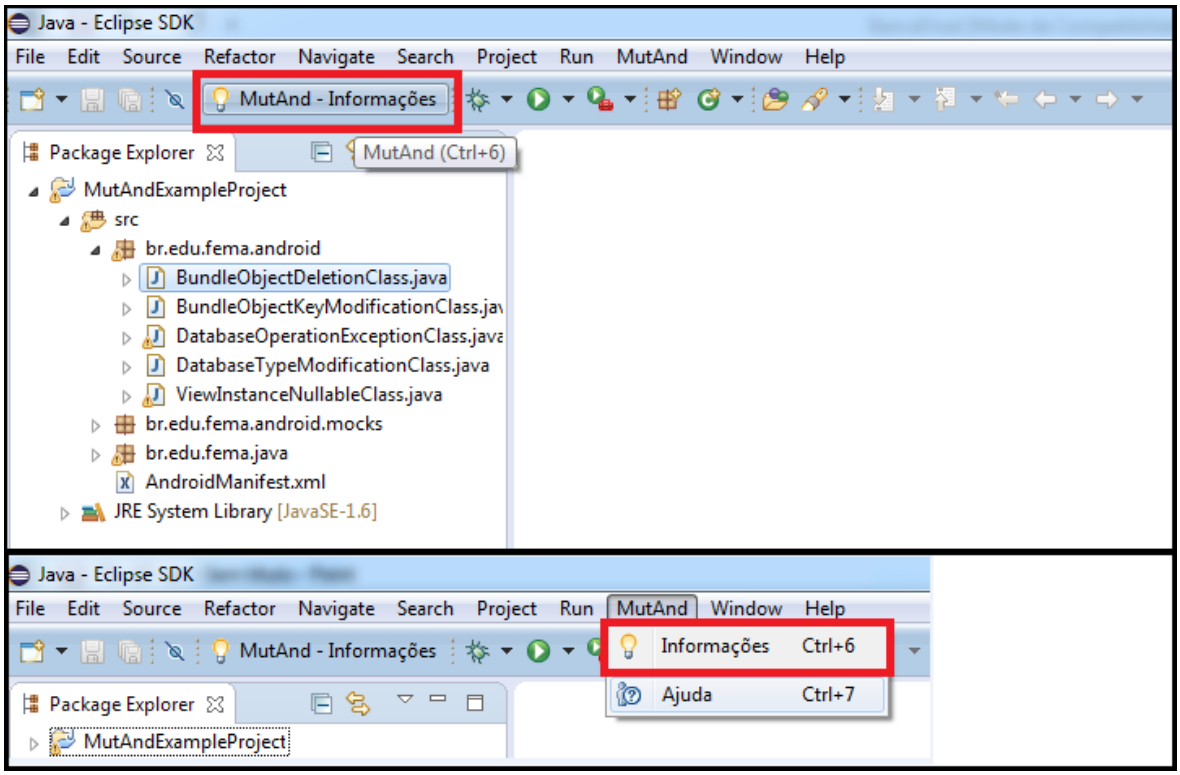

Figura 46 – Método de verificar as informações do MutAndAPI

Quando a opção de informações é aberta, a Figura 47 exemplifica o ocorrido.

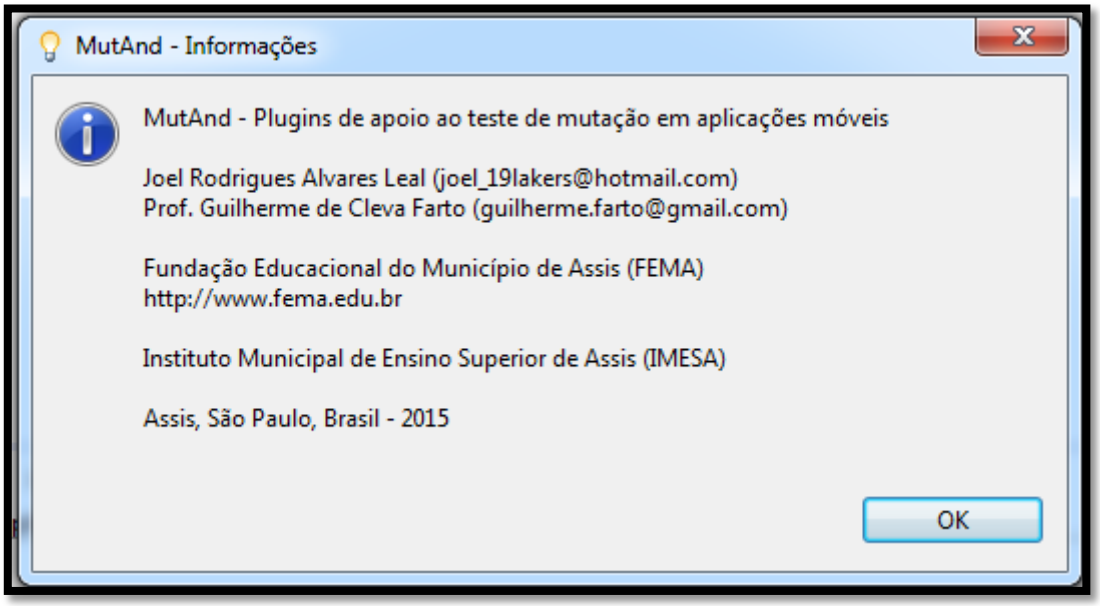

Figura 47 – Janela de informações aberta

Para acessar a ajuda do *plugin* existe duas opções, no menu escolher a opção MutAnd e logo em seguida selecionar Ajuda ou no menu Help selecionar a opção *Help Contents*. A Figura 48 exemplifica como abrir a ajuda do MutAndAPI.

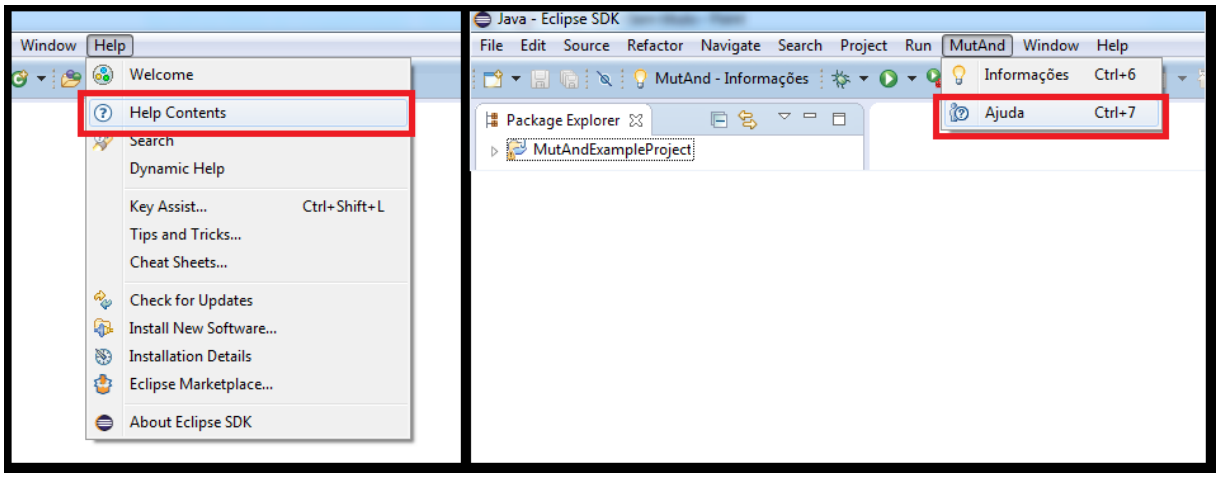

Figura 48 – Como abrir ajuda

Quando a ajuda é aberta, existe várias opções de ajuda, tem que ser selecionado o tópico MutAnd – *Plugins* de apoio do teste de mutação, dentro deste tópico tem vários outros sub tópicos para o aprendizado do teste de mutação, sendo eles, Teste de Mutação, Operadores de Mutação, Mutantes e Operadores de Mutação MutAnd. Dentro do Operadores de Mutação MutAnd existe duas opções, Operadores de Mutação Java e Operadores de Mutação em *Android*. A Figura 49 exemplifica a ajuda do MutAndAPI aberta com suas informações.

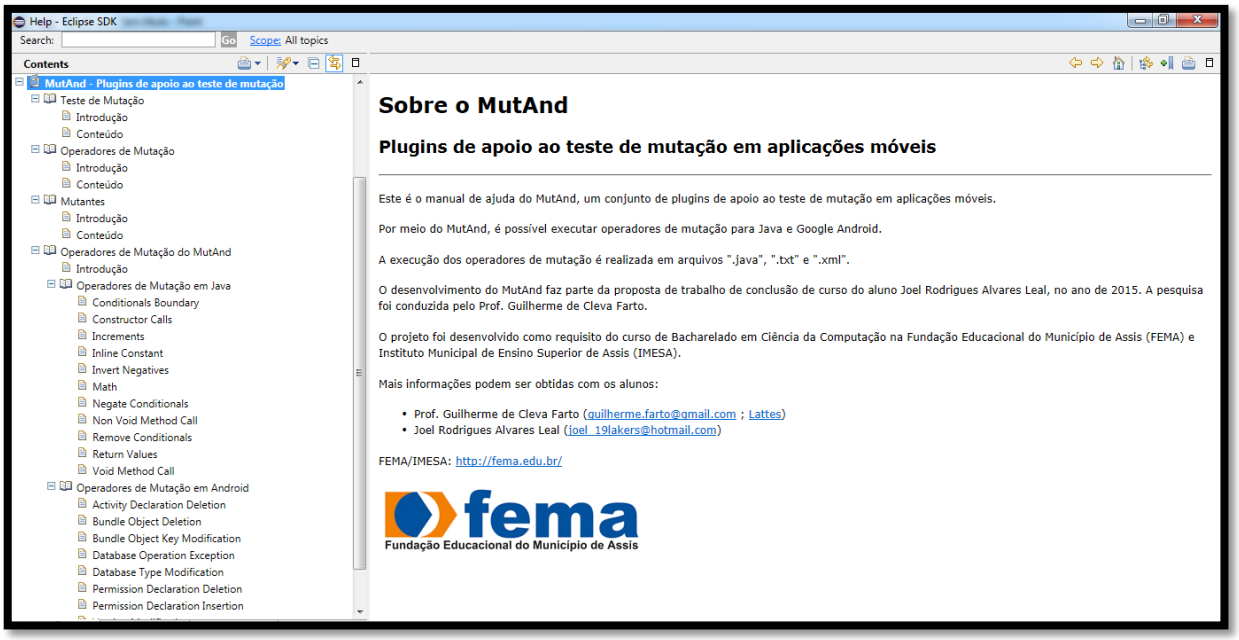

Figura 49 – Ajuda aberta

Para realizar a mutação em algum arquivo ou pacote é necessário clicar com o botão direito no mesmo, ir até a opção MutAnd e depois selecionar a opção Operadores de Mutação, a Figura 50 é ilustrada a explicação.

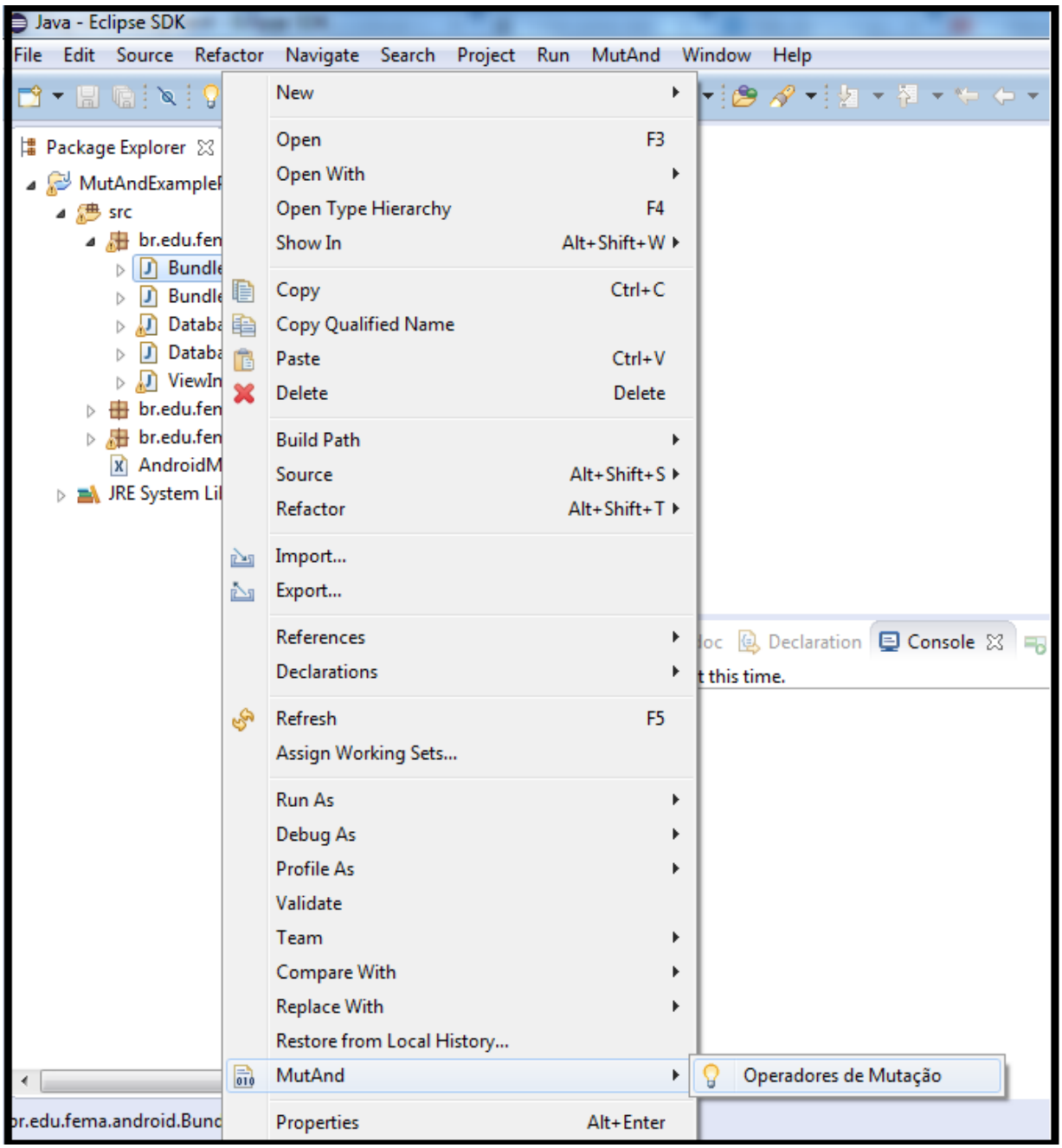

Figura 50 – Abrindo o MutAndPopUp

Logo após escolher está opção abrirá uma janela que é possível escolher os operadores de mutação Java para ser executado, para escolher qual mutação será realizada tem que ser selecionado a opção, a Figura 51 exemplifica tal tela.

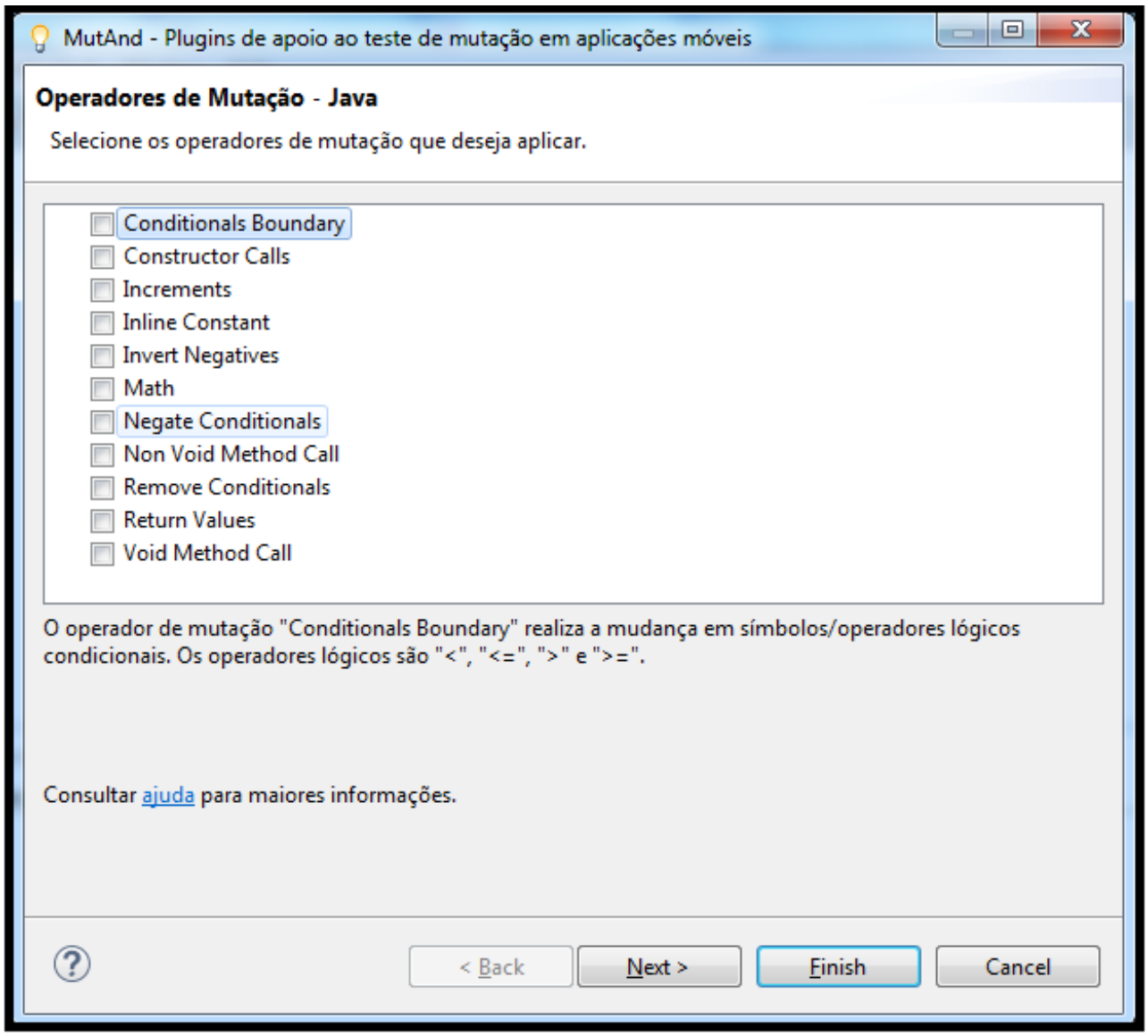

Figura 51 – Tela de seleção dos operadores de mutação Java

Logo em seguida abre outra janela é possível escolher os operadores de mutação *Google Android* que será executado para a mutação de tal arquivo, como na opção anterior, tem que selecionar quais operadores de mutação *Google Android* será feito a modificação, a Figura 52 ilustra esta tela.

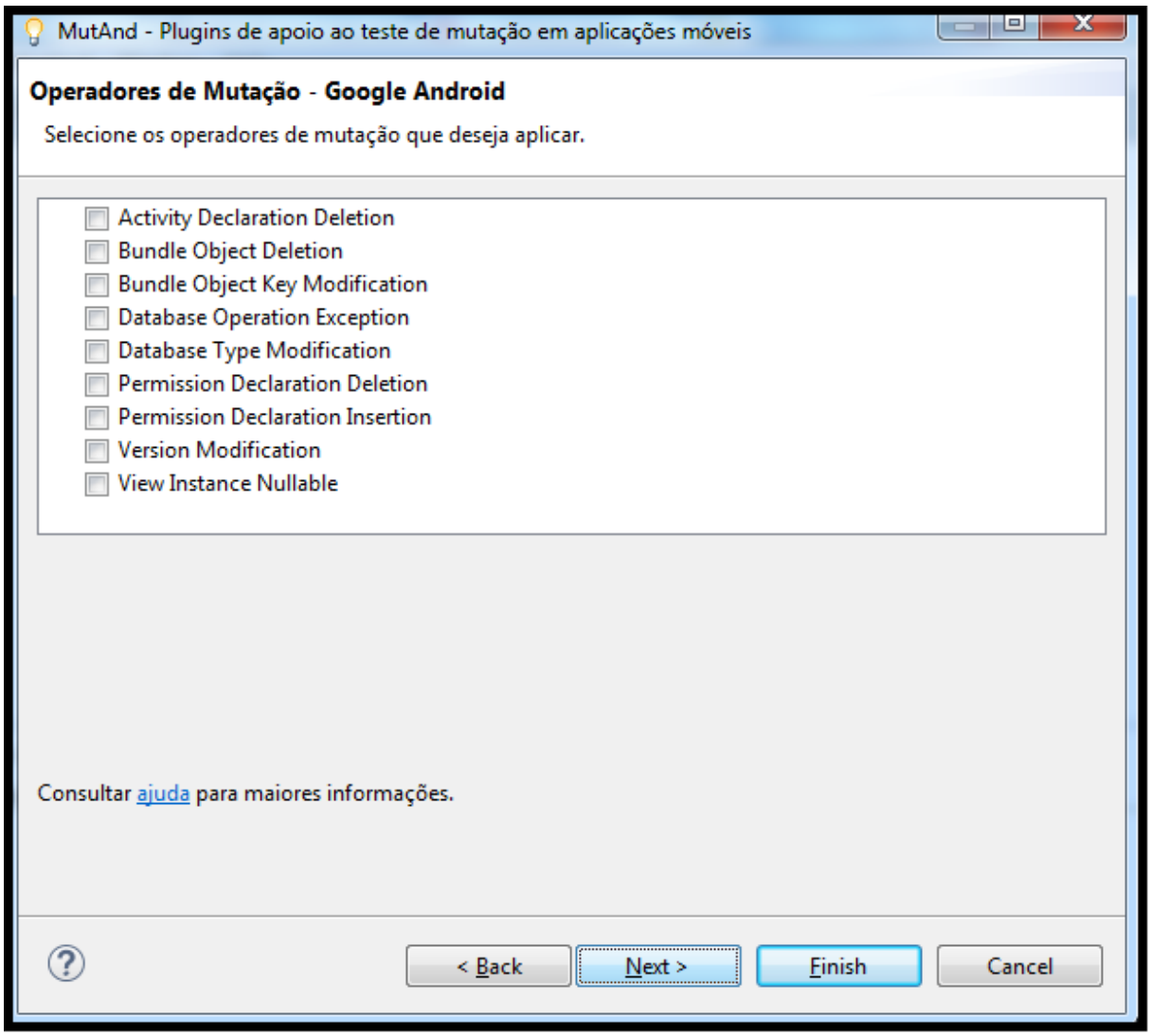

Figura 52 – Tela de seleção dos operadores de mutação *Google Android*

Nas opções de operadores de mutação Java e *Google Android,* em caso de não ser selecionado nenhum operador não tem nenhum problema, pois as vezes precisa ser executado apenas operadores Java ou apenas operadores *Google Android.*

Logo após a seleção dos operadores que serão executados a próxima etapa é a escolha do diretório de saída e quais são os extras em caso de operadores *Google Android*, porque quando é necessário deletar alguma *activity* especifica, dar permissão ou remover permissão especifica é necessário informar qual será, então é essa a função do extra, armazenas os valores específicos para certos operadores. A Figura 53 ilustra o que foi descrito anteriormente.

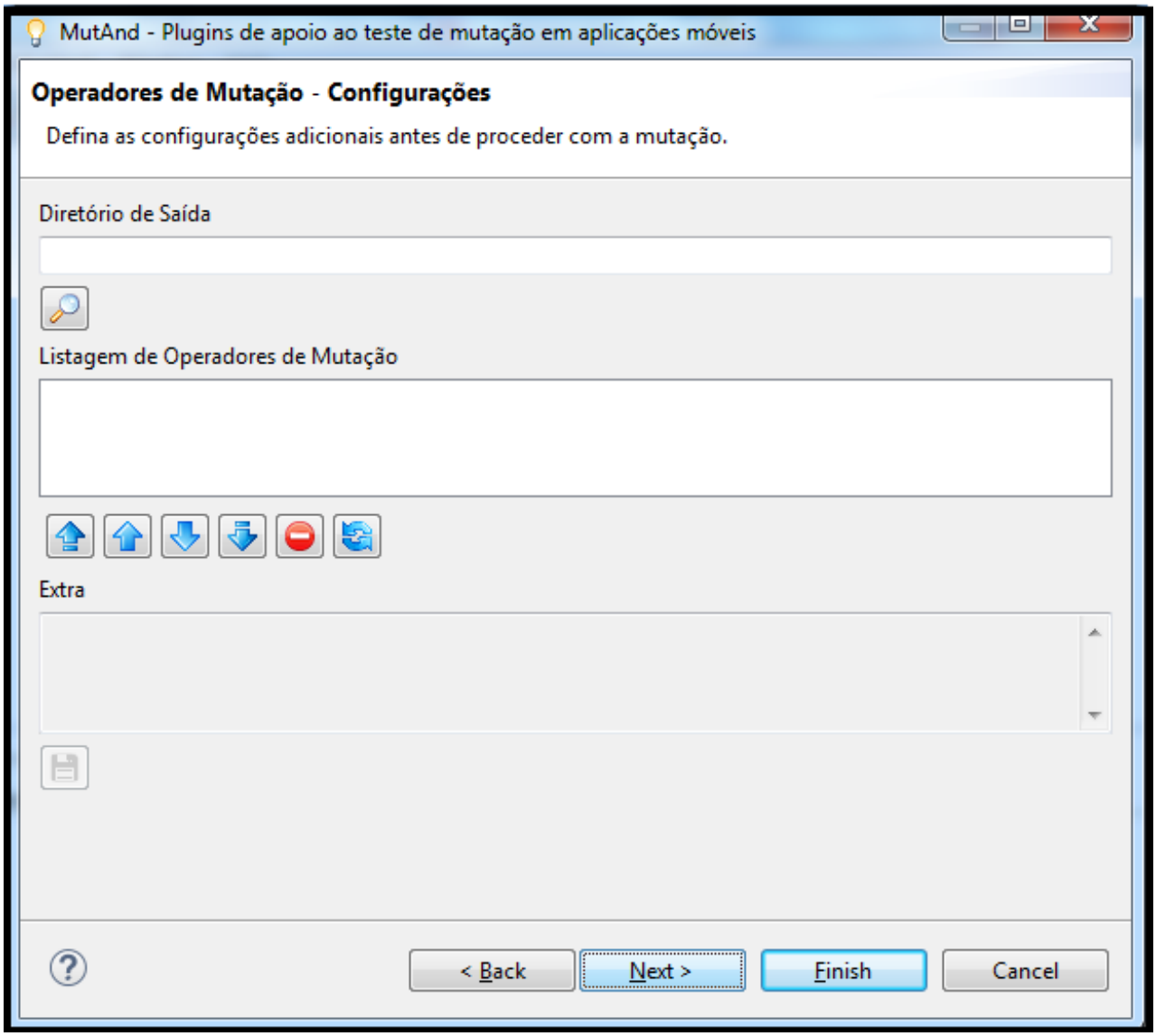

Figura 53 – Tela onde é selecionado o diretório de saída e os valores extras

Logo após a seleção do diretório de saída e a inserção do extra tem uma tela na qual é para confirmar as classes que serão modificadas, basta deixar selecionado o que é para ser modificado e remover a seleção do que não é para ser modificado. A Figura 54 ilustra tal tela.

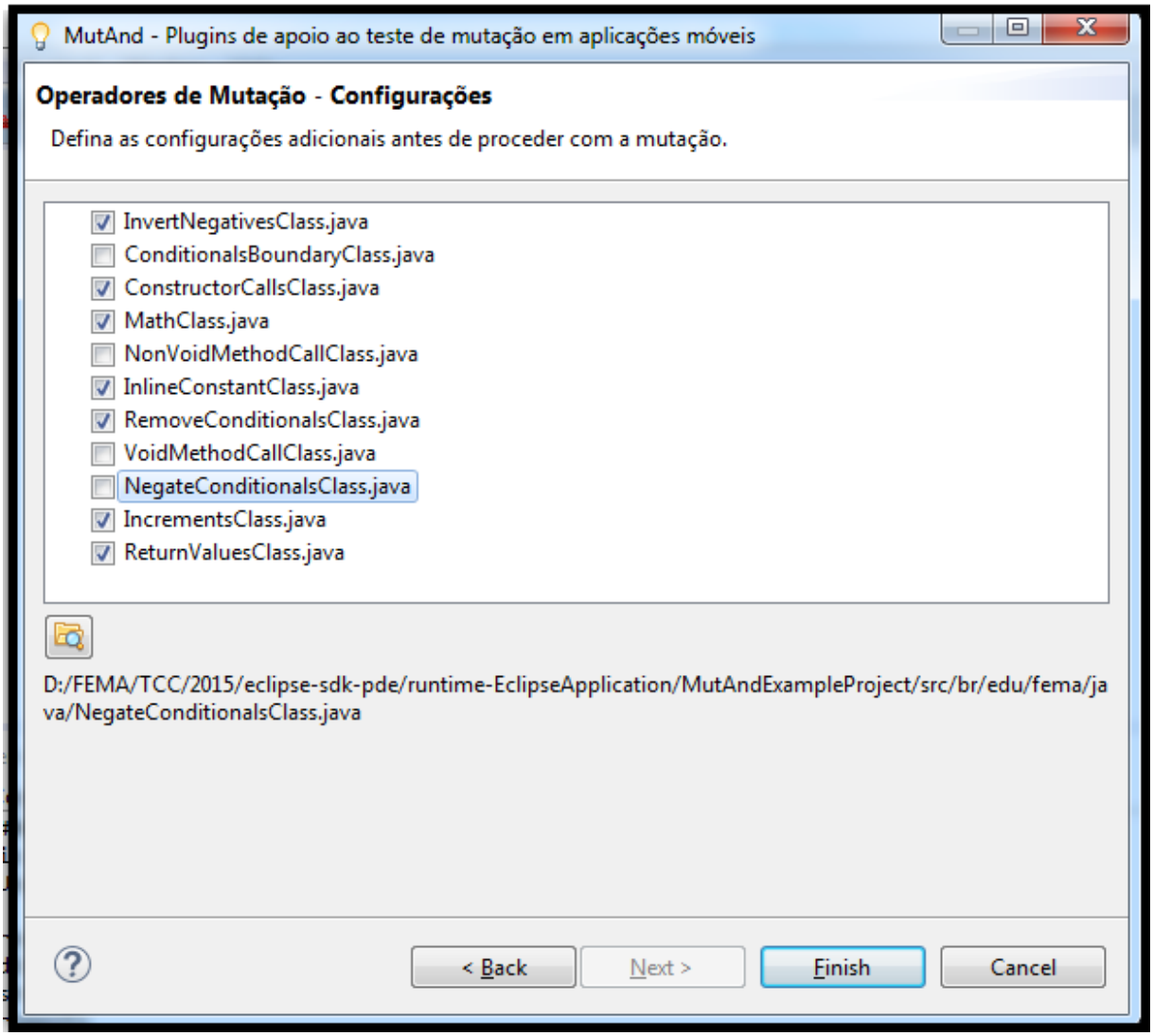

Figura 54 – Tela onde é para selecionar as classes que serão realizadas a mutação

Após o término da mutação, no diretório escolhido estará o arquivo modificado, a Figura 55 ilustra o código fonte original e a Figura 56 mostra o código fonte modificado após o teste de mutação *MathMutator.*

```
package br.edu.fema.java;

public class MathClass {

	public static void main(String[] args) {
		int a = 3;
```

```
		int b = 5;
		int c = 10;
		int d = 15;

		a = b + c;

		b = a - b;

		c = a / b;

		d = a * b;

		a = a << b;
		b = b >>> a;
	}

}
```

Figura 55 – *MathClass* antes de ser modificada

```
package br.edu.fema.java;

public class MathClass {

	public static void main(String[] args) {
		int a = 3;
		int b = 5;
		int c = 10;
		int d = 15;

		a = b - c;

		b = a + b;

		c = a * b;

		d = a / b;

		a = a >> b;
		b = b <<< a;
	}

}
```

Figura 56 – *MathClass* depois de ser modificado

A Figura 57 ilustra o arquivo *AndroidManifest.xml* original sem ser modificado.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
	package="br.edu.fema.sistemaacademico" android:versionCode="1"
	android:versionName="1.0">
```

```
	<uses-sdk
		android:minSdkVersion="10"
		android:maxSdkVersion="20"
		android:targetSdkVersion="10" />

	<application android:allowBackup="true" android:icon="@drawable/ic_launcher"
		android:label="@string/app_name" android:theme="@style/AppTheme">
		<activity android:name=".view.PrincipalActivity">
			<intent-filter>
				<action android:name="android.intent.action.MAIN" />

				<category
android:name="android.intent.category.LAUNCHER" />
			</intent-filter>
		</activity>

		<activity android:name=".view.CursosListActivity">
		</activity>

		<activity android:name=".view.CursoActivity">
		</activity>

		<activity android:name=".view.AlunosListActivity">
		</activity>

		<activity android:name=".view.AlunoActivity">
		</activity>
	</application>

	<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

</manifest>
```

Figura 57 – Arquivo XML original

A figura 58 ilustra o código fonte do arquivo *AndroidManifest.xml* modificado, nele foi utilizado o operador de mutação *ActivityDeclarationDeletionMutator.*

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
	package="br.edu.fema.sistemaacademico" android:versionCode="1"
	android:versionName="1.0">

	<uses-sdk
		android:minSdkVersion="10"
		android:maxSdkVersion="20"
		android:targetSdkVersion="10" />

	<application android:allowBackup="true" android:icon="@drawable/ic_launcher"
		android:label="@string/app_name" android:theme="@style/AppTheme">
		<activity android:name=".view.PrincipalActivity">
			<intent-filter>
				<action android:name="android.intent.action.MAIN" />
```

```
                        <category
android:name="android.intent.category.LAUNCHER" />
                    </intent-filter>
            </activity>

            <activity android:name=".view.CursosListActivity">
            </activity>

            <activity android:name=".view.CursoActivity">
            </activity>

            <activity android:name=".view.AlunosListActivity">
            </activity>

            <activity android:name=".view.AlunoActivity">
            </activity>
      </application>

</manifest>
```

Figura 58 – AndroidManifest.xml depois de ser modificado com o operador de mutação *ActivityDeclarationDeletionMutator*

A Figura 59 ilustra um código fonte original na qual será realizado a mutação com o operador *ConditionalsBoundaryMutator.*

```
package br.edu.fema.java;

public class ConditionalsBoundaryClass {

      public static void main(String[] args) {
            int a = 5;
            int b = 10;

            if (a < b) {
                  // Código
            }

            if (a <= b) {
                  // Código
            }

            if (a > b) {
                  // Código
            }

            if (a >= b) {
                  // Código
            }
```

```
	}

}
```

Figura 59 – Código fonte original

A Figura 60 ilustra o código fonte da Figura 59 logo depois de ser modificado pelo operador de mutação *ConditionalsBoundaryMutator.*

```
package br.edu.fema.java;

public class ConditionalsBoundaryClass {

	public static void main(String[] args) {
		int a = 5;
		int b = 10;

		if (a <= b) {
			// Código
		}

		if (a < b) {
			// Código
		}

		if (a >= b) {
			// Código
		}

		if (a > b) {
			// Código
		}
	}

}
```

Figura 60 – Código fonte modificado após a mutação *ConditionalsBoundaryMutator*

6.7 CONSIDERAÇÕES FINAIS

A API e os *plugins* propostos e desenvolvidos foram executados e validados com sucesso, mostrando-se uma possibilidade viável para o teste de mutação. Dentro da API desenvolvida é possível incorporar novos *plugins* à plataforma proposta do

MutAndAPI, bem como o desenvolvimento e novos operadores de mutação para Java e *Google Android.*

7. CONCLUSÃO

No desenvolvimento do projeto foi estudado teste de software, teste de mutação, tecnologia *Google Android,* tecnologia Java, Eclipse PDE, entre outros estudos, para que ser possível o desenvolvimento do *plugin* MutAndAPI.

MutAndAPI é o produto final desse projeto, no qual tem os operadores de mutação para Java e *Google Android*, e também tem *plugins* baseado na plataforma Eclipse para apoiar o teste de mutação.

A utilização do Eclipse PDE tem alguns vantagens em relação ao Eclipse IDE que é utilizado para o desenvolvimento de software, entre as vantagens que tem podem ser destacadas o desenvolvimento de *plugins,* criação de novas funcionalidades, extensão dos recursos existentes e facilidade na elaboração de novas ferramentas baseada no Eclipse.

Com o desenvolvimento do *plugin* MutAndAPI percebe-se que a possibilidade de melhoria nos testes de aplicações Java e *Google Android* se torna cada vez melhor, uma vez que o teste de mutação é aplicado mais facilmente através do *plugin* proposto*,* assim facilitando, agilizando e tendo um histórico dos testes realizados.

É de extrema importância o desenvolvimento de novos componentes para o Eclipse pois facilita o dia-a-dia de pessoas que trabalham com Java, *Google Android,* bem como qualquer outro contexto onde novas funcionalidades podem ser acrescentadas à IDE do Eclipse.

7.1 LIMITAÇÕES DA PESQUISA

O objetivo do trabalho e escopo se limitam ao estudo, proposta, desenvolvimento e execução dos operadores de mutação e *plugins* desenvolvidos para apoiar o teste de mutação. Portanto, não foi definida uma avaliação experimental para executar e validar os arquivos e aplicação Java ou *Google Android* após a execução dos operadores de mutação.

7.2 TRABALHOS FUTUROS

A partir deste projeto é possível dar continuidade, desenvolvendo novos *plugins* e incorporando novos operadores de mutação no MutAndAPI, visto que os primeiros passos já foram dados com a realização de tal projeto.
Para trabalhos futuros pode-se destacar:

- Reincorporação dos códigos fontes mutados (modificados) ao projeto original
- Controle e versionamento dos códigos fontes mutados
- Desenvolvimento de novos *plugins* para estender as funcionalidades existentes como, por exemplo, comparação de versões anteriores, restaurar para outra versão e uma árvore gráfica que demonstre os operadores de mutação aplicados em arquivos ou em todo um projeto Java ou *Google Android.*

REFERÊNCIAS

AQUINO, Juliana França Santos. **Plataformas de Desenvolvimento para Dispositivos Móveis.** Programa de Pós Graduação em Informática. Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio. 14p. Rio de Janeiro. Rio de Janeiro. 2007.

ASCARI, Luciano César. **Teste Baseado em Defeitos de Classes Java Utilizando Aspectos e Mutação de Especificações OCL.** 2009. 93p. Dissertação Mestrado – Programa de Pós-Graduação em Informática, Setor de Ciências Exatas – Universidade Federal do Paraná, Paraná, Curitiba, 2009.

BALSIGER, Markus. **A Quick-Start Tutorial to Eclipse *Plugin* Development.** 2010. 24p. Institut fur Informatik und angewandte Mathematik. 2010.

BARBOSA, Ellen Francine; MALDONADO José Carlos; VINCENZI, Auri Marcelo Rizzo; DELAMARO, Márcio Eduardo; SOUZA, Simone do Rocio Senger; JINO, Mario. **Introduçao ao teste de software.** 2000. 49p. Minicurso apresentado no XIV Simpósio Brasileiro de Engenharia de Software (SBES 2000), Universidade de São Paulo (ICMC/USP), São Paulo, São Paulo, 2000.

BEIZER, Boris. **Software Testing Techniques.** Van Nostrand Reinhold Company, New York, 2nd edition, 1990.

BINDER, Robert V. Testing Object-Oriented Systems: Models, Patterns, and Tools, volume 1. Addison-Wesley LongMan, Inc., 1999.

CAMPANHA, Diogo Nascimento. **Teste de mutação nos paradigmas procedimental e oo: uma avaliação no contexto de estrutura de dados.** 2010. 94p. Dissertação Mestrado – Instituto de ciência da Computação – ICMC/USP, São Paulo, São Carlos, 2010.

DANTAS, Valéria Lelli Leitão. **Requisitos para Teste de Aplicações Móveis.** 2009. 131. Monografia (Mestrado) – Departamento de Computação – Universidade Federal do Ceará, Ceará, Fortaleza, 2009.

DEITEL, H.M.: **Android for Programmers,** Indiana - USA. 2012.

DEMILLO, Richard. Software testing and evaluation. The benjamim/Commings Publishing Company, Inc, 1978.

DEMILLO, Richard; OFFUTT, Jefferson A. Constraint-Based Automatic Test Data Generation. IEEE Trans. Softw. Eng., 17(9):900_910, 1991.

DEMILLO, Richard; LIPTON, Richard. J.; SAYWARD, Frederick. G. Hints on test data selection: Help for the practicing programmer. IEEE Computer, 11(4):34.43, April 1978.

DEUS, Gilcimar Divino de. **Avaliação de Técnicas de Teste para Dispositivos Móveis por Meio de Experimentação.** 2009. 117p. Monografia (Mestrado) – Instituto de Informática - Universidade Federal de Goiás, Goiás, Goiânia, 2009.

ENDEL1. **Mundo terá quase 7 bilhões de celulares em uso até o final de 2014, diz estudo.** Disponível em: <http://veja.abril.com.br/noticia/vida-digital/mundo-tera-quase-7-bilhoes-de-celulares-em-uso-ate-o-final-de-2014-diz-estudo/>. Acesso em: 27 de fevereiro de 2015.

ENDEL2. **Google Play.** Disponível em: <https://play.google.com/store>. Acesso em 03/03/2015.

ENDEL3. **Extending the Eclipse IDE - *Plugin* development – Tutorial.** Disponível em: <http://www.vogella.com/tutorials/Eclipse*Plugin*/article.html>. Acesso em 03/03/2015.

ENDEL4. **Eclipse.** Disponível em: <https://eclipse.org/pde/>. Acesso em 03/03/2015.

ENDEL5. **Top 10 *Plugins* for 2013.** Dispónivel em: <http://www.eclipse.org/community/eclipse_newsletter/2013/december/article2.php>. Acesso em 08/03/2015.

ENDEL6. **Android.** Disponível em: <http://www.android.com/tv/>. Acesso em 10/03/2015.

ENDEL7. **Android.** Disponível em: <http://www.android.com/wear/>. Acesso em 10/03/2015.

ENDEL8. **Android.** Disponível em: <http://www.android.com/auto/>. Acesso em 10/03/2015.

ENDEL9. **The Mutation Process.** Disponível em: <http://muclipse.sourceforge.net/about.php>. Acesso em 21/03/2015.

ENDEL10. **DigiCal+ Calendário.** Disponível em: <https://play.google.com/store/apps/details?id=com.digibites.calendarplus>. Acesso em 21/03/2015.

ENDEL11. **Mind Tools.** Disponível em: <https://play.google.com/store/apps/details?id=com.mindtools>. Acesso em 21/03/2015.

ENDEL12. **Google Drive.** Disponível em: <https://play.google.com/store/apps/details?id=com.google.android.apps.docs&hl=pt_BR>. Acesso em 21/03/2015.

ENDEL13. **Obtenha Informações sobre a Tecnologia Java.** Disponível em: <https://www.java.com/pt_BR/about/>. Acesso em 21/03/2015.

EXAME. **Mercado de aplicativos móveis vai crescer explosivamente, prevê IDC.** São Paulo. Disponível em <http://exame.abril.com.br/tecnologia/noticias/mercado-de-aplicativos-moveis-vai-crescer-explosivamente-preve-idc>. Acesso em 28/10/2014.

FABBRI, Sandra C P F; VINCENZI, Auri Marcelo Rizzo; MALDONADO, José Carlos. **Introdução ao teste de software**. 1 edição, 2007.

FALBO, Ricardo de Almeida. **Engenharia de Software.** 2005. 99p. Universidade Federal do Espírito Santo. 2005.

FARIA, Alessandro de Oliveira. **Programe se androide.** Linux Magazine. Volume 1. Número 43. 73-77p. 2008.

LECHETA, Ricardo R. **Google Android: Aprenda a cirar aplicações para dispositivos moveis Android SDK.** 3 ed São Paulo. Novatec, 2013. 576p.

MA, Yu-Seung; OFFUTT, Jeff. **Description of Class Mutation Mutation Operators for Java.** 2014.

LINKMAN, S.; VINCENZI, Auri Marcelo Rizzo; MALDONADO, José Carlos. **An evaluation of systematic functional testing using mutation testing.** 7 International Conference on Empirical Assessment in Software Engineering – EASE. The IEE, abril 2003.

MA, Yu-Seung; OFFUTT, Jeff. **Description of Method-level Mutation Operators for Java.** 2005.

MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom; THOMAS, Todd M. **The Art of Software Testing.** John Wiley & Sons, 2nd. edition, 2004.

NOBRE Tiago; VERGILIO Silvia R.; POZO Aurora T. R. **Reduzindo o Custo de Teste de Mutação de Interface com Algoritmos de Otimização Multi-objetivo.** 2012. 12p. Departamento de Informática. Universidade Federal do Paraná, Paraná, Curitiba, 2012.

PEREIRA, Lucio Camilo de Oliveira; SILVA, Michel Lourenço da. **Android para desenvolvedores.** Rio de Janeiro: Brasport, 2009. 221 p.

PRESSMAN, Roger S. **Engenharia de Software – Uma Abordagem Profissional.** McGraw-Hill. 2006.

SOMMERVILLE, Ian. **Engenharia de software.** 6 edição. Addison Wesley. 2003.

TONIN, Graziela Simone. **Tendências em Computação Móvel.** 2012. 17p. Departamento de Ciência da Computação. Universidade de São Paulo. São Paulo. São Paulo, 2012.