Teste de mutação nos paradigmas procedimental e oo: uma avaliação no contexto de estrutura de dados

Diogo Nascimento Campanha

Teste de mutação nos paradigmas procedimental e oo: uma avaliação no contexto de estrutura de dados

Diogo Nascimento Campanha

Orientador: ***Prof. Dr. José Carlos Maldonado***

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC/USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Outubro/2010

Agracimentos

Agradeço muito a Deus por todos os momentos de esperança e por proteger e guiar minha vida até aqui.

A toda minha família pelo amor, apoio e auxílio na vinda para São Carlos. Em especial a minha esposa Vanessa, meus pais Alberto e Marli e minha irmã Daniella, sem vocês o mestrado não seria possível.

Ao meu orientador José Carlos Maldonado e minha co-orientadora Simone do Rocio Senger de Souza, pela oportunidade, profissionalismo, incentivo, orientação e paciência, sem o qual não teria realizado este trabalho.

Muito obrigado aos meus amigos do LabES e agregados: Fabiano, Van, Maria, Jorge Piu, Otávio, Rodolfo, Vânia, Marcelo, Endo, Abe, Paulo Ceará, Sandro KLB, Mel, Erika, Nerso, Messias, Ana Carla, Marcão, Katia, Marllos, Sorin, David, Silvana, Lucas Bueno, Rafael, Adriano, Vinicius, Andrézinho, Bruno Cafeo, Draylson, Nardi, Bruno Feres, José Tim e as Profs. Ellen Francine e Rosana Braga. Obrigado a todos!

Aos amigos de Campo Grande e da república Tereré em São Carlos pelos momentos de alegria e por tornar a estadia em São Carlos mais agradável, Gondim, Mario, Kenji, Alex, Alessandro, Patrick, Letrícia, Márcio, Jucimara, Maxwell, Kish, Lucas, Ronaldo. Obrigado de coração!

À Fapesp e CNPq pelo apoio financeiro.

Resumo

Com o objetivo de auxiliar a definição e evolução de estratégias de testes, estudos experimentais vêm sendo realizados comparando diferentes técnicas e critérios de teste em relação ao custo, eficácia e dificuldade de satisfação (*strength*). Entretanto poucos estudos buscam avaliar os critérios em diferentes paradigmas. Esta avaliação é importante pois o paradigma de implementação influência significativamente no programa gerado e as características entre programas implementados em diferentes paradigmas pode influenciar em diversos aspectos da atividade de teste. Este estudo é complementar a um outro trabalho do grupo do laboratório de engenharia de software do ICMC em que foram comparados o custo da aplicação dos critérios da técnica Estrutural em relação aos paradigmas Procedimental e Orientado a Objetos. Este trabalho apresenta um estudo experimental comparando o custo e o *strength* do critério Análise de Mutantes nos dois paradigmas. Além da avaliação do critério Análise de Mutantes, o material gerado para este estudo será construído de forma que possa ser utilizado para o ensino e treinamento das principais técnicas e critérios de teste e espera-se que este possa contribuir de alguma forma para que o ensino de teste de software possa ser aplicado em paralelo com o ensino de algoritmos e estrutura de dados. Para a condução deste estudo, foi utilizado um conjunto de 32 programas do domínio de estrutura de dados com versões implementadas em C e em Java. O critério Análise de Mutantes foi aplicado com auxílio das ferramentas Proteum e MuClipse. Para a avaliação do *strength*, o conjunto de casos de teste adequado a um programa foi executado contra os mutantes gerados na mesma versão do programa implementado no outro paradigma de interesse e o escore de mutação avaliado (*cross scoring*). Resultados indicam que tanto o custo quanto o *strength* do teste de mutação é maior em programas implementados no paradigma Procedimental do que no paradigma OO. Resultados estes certamente influenciados pelo conjunto de operadores implementado nas duas ferramentas. No paradigma procedimental, também foi avaliado o escore de mutação

obtido por um subconjunto dos operadores da Proteum, construído com o objetivo de reduzir o custo da aplicação do critério. O escore obtido foi satisfatório e as reduções no custo significativas. Também foi avaliado *strength* das técnicas Funcional e Estrutural em relação ao critério AM nos dois paradigmas. Os resultados mostram que os conjuntos de casos de teste adequados aos critérios das técnicas Funcional e Estrutural no paradigma OO obtiveram, em geral, um escore de mutação maior do que no paradigma Procedimental.

Abstract

Aiming at to assist the definition and evolution of testing strategies, empirical studies have been conducted comparing the test criteria in terms of cost effectiveness and strength. However few studies were conducted comparing the criteria in different paradigms. This assessment is important because program paradigm can cause a big influence in the result program and the characteristics of this programs can cause influences in testing activities. This study complements another work in this group that compared the cost of Structural test technique considering the Procedural and Object Oriented paradigms. This paper presents an experimental study comparing the cost and strength of Mutation Test in Procedural and OO paradigms. Besides the evaluation of mutation test, the results of this study can be used to help teaching and training many test criteria and it may somehow contribute to teach software testing n parallel with algorithms and data structures programs. It was used a set of 32 programs of data structure programs with versions in C and Java. For this study it were used Proteum and MuClipse to execute the mutation tests. It was also evaluated the strength of a set of test cases (TC) appropriate to a program in a paradigm in the same version of the program implemented in the other paradigm of interest (cross scoring). Results indicate that both cost and strength is higher in Procedural programs than in OO. This results are certainly influenced by the set of operators built in each tool studied. It was also measured the mutation score of a subset of Proteum's operators built to reduce the cost of Mutation Testing in Procedural programs. The mutation score obtained was satisfactory and there was a significant reduction in cost. It was also evaluated the strength of Functional and Structural techniques in Mutation Test for both paradigms. The results show that the mutation score of the set of test cases adequate to Functional and Structural techniques were in general, higher in OO paradigm than in Procedural paradigm.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Programas

CAPÍTULO

1

Introdução

1.1 Contexto e Motivação

Com o avanço da tecnologia e a diminuição dos custos de hardware os dispositivos computacionais estão cada vez mais presentes no cotidiano. Em conjunto com essa expansão, aumentou também a preocupação com a qualidade do processo de produção e do produto construído. Atividades de Verificação, Validação e Teste (VV & T) não se restringem ao produto final. Ao contrário, podem e devem ser conduzidas durante todo o processo de desenvolvimento de software, desde sua concepção (Delamaro *et al.*, 2007b).

Dentre as atividades de VV & T, Delamaro *et al.* (2007b) descrevem o teste de software como uma atividade dinâmica cujo intuito é executar o programa ou modelo utilizando algumas entradas em particular e verificar se seu comportamento está de acordo com o esperado. É importante ressaltar que ao testar um programa não é possível garantir que o mesmo não possua erros, mas sim aumentar a probabilidade de que os erros sejam detectados caso existam (Myers *et al.*, 2004). Segundo Delamaro *et al.* (2007b), idealmente o programa deveria ser testado com todos os valores possíveis do domínio de entrada. Essa atividade porém é na maioria das vezes impraticável devido à cardinalidade do domínio de entrada. Assim, técnicas e critérios de teste são propostos com o objetivo de selecionar um subconjunto do domínio de entrada com maior probabilidade de identificar os defeitos existentes no programa, caso estes existam.

Além de permitir que a atividade de teste possa ser conduzida de maneira sistemática, as técnicas de teste fornecem informações sobre como testar o software, quando parar os testes e, se possível, fornecem uma medida objetiva do nível de confiança e de qualidade alcançados com os testes realizados (DeMillo, 1980). Em geral, os critérios de teste de software são estabelecidos a partir das técnicas: Funcional, Estrutural e Baseada em Erros. Na técnica Funcional os requisitos são derivados da especificação do programa. Na técnica Estrutural o código-fonte é avaliado para a determinação dos requisitos. A técnica de teste Baseada em Erros utiliza dados sobre os erros cometidos por programadores durante o desenvolvimento de software para definir os requisitos de satisfação (Maldonado *et al.*, 1998).

O critério Análise de Mutantes tem sido investigado por diversos pesquisadores sob diferentes aspectos, principalmente devido a sua flexibilidade para ser aplicado em diversos artefatos de software. O critério utiliza um conjunto de programas ligeiramente modificados (mutantes) obtidos a partir de determinado programa P para avaliar o quanto um conjunto de casos de teste T é adequado para o teste de P. O objetivo é determinar um conjunto de casos de teste que consiga revelar, por meio da execução de P, as diferenças de comportamento existentes entre P e seus mutantes (DeMillo, 1978).

Apesar das técnicas e critérios definidos permitirem a aplicação sistemática dos testes. Essa atividade pode representar um custo alto dependendo da complexidade do projeto. Com o objetivo de reduzir os custos dessa atividade e manter a alta probabilidade de identificar os erros, estratégias de testes devem ser exploradas combinando as técnicas e critérios e assim explorar as vantagens de cada técnica de maneira complementar.

Para a definição de uma boa estratégia de teste além da escolha da combinação dos critérios, deve ser considerado também o paradigma no qual o software deve ser implementado. O paradigma de desenvolvimento influencia na abstração do problema e fornece a base para a representação das estruturas. Dessa forma, programas implementados em diferentes paradigmas tendem a apresentar uma organização e estruturação do código-fonte diferentes. Segundo Vincenzi *et al.* (2007), o uso da orientação a objetos não é capaz de garantir o correto funcionamento de um programa por si próprio. Pelo contrário, o que se tem observado é que as características principais da orientação a objetos introduzem novas fontes de falhas de modo que atividades de VV&T são de fundamental importância no contexto de programas orientado a objetos.

A partir de estudos experimentais na área de teste de software é possível avaliar os critérios considerando aspectos como o custo de aplicação, eficácia em revelar defeitos e dificuldade de satisfação (*strength*) (Maldonado *et al.*, 1998). Ao avaliar os critérios de testes em diferentes paradigmas de programação uma base de conhecimento pode ser construída possibilitando,

assim como já acontece em paradigmas consolidados como o caso do paradigma procedimental, a determinação de estratégias de testes mais eficientes com baixo custo de aplicação.

Para que a atividade de testes seja realizada com sucesso, além de uma estratégia bem definida, os testes devem ser conduzidos por profissionais com conhecimento e experiência na área. Para isso é importante a disponibilidade de materiais que auxiliem a formação e capacitação desses profissionais. De acordo com Myers *et al.* (2004) a pesquisa de técnicas de teste de software amadureceu bastante nos últimos anos, porém esforços devem ser somados tanto pela indústria quanto pela academia para que a aplicação dos testes seja incorporada com mais vigor no processo de desenvolvimento de software. Em seu trabalho, o autor identifica alguns obstáculos que prejudicam o ensino de teste de software. O autor defende a idéia de que os estudantes devem ser apresentados às técnicas de software juntamente com o aprendizado da programação, pois assim, o aluno pode criar o hábito de desenvolver os programas e testá-los de forma sistemática.

Um dos obstáculos para o ensino dessas duas atividades em conjunto é o custo para a configuração e preparação do material necessário ao ensino prático das técnicas de teste. Um ambiente integrado contendo ferramentas, manuais e documentação de como aplicar os principais critérios existentes pode auxiliar esse aprendizado, contribuindo assim para que o aluno concentre-se mais na aplicação correta das técnicas e menos com a configuração e funcionamento das ferramentas.

1.2 Objetivo

Este trabalho tem como objetivo avaliar o custo e a dificuldade de satisfação (*strength*) do critério Análise de Mutantes (AM) entre o paradigma Procedimental e Orientado a Objetos (OO). Com o material gerado neste estudo, pretende-se também construir um conjunto de artefatos que possa ser utilizado para auxiliar o ensino e treinamento das principais técnicas e critérios de testes.

Para alcançar este objetivo foi conduzido um estudo experimental aplicado a um *benchmark* de programas do domínio de estrutura de dados com especificações equivalentes, implementados nos dois paradigmas.

Os programas utilizados e os resultados gerados em outro trabalho de mestrado (Prado, 2009) foram considerados para o desenvolvimento deste projeto. Em seu trabalho, Prado (2009) comparou o custo e a dificuldade de satisfação dos critérios baseados em Fluxo de Dados e Fluxo de Controle na técnica Estrutural no contexto de programas de estruturas de dados Procedimentais e OO.

Outro objetivo do trabalho é analisar e comparar o *strength* dos critérios das técnicas Funcional e Estrutural em relação ao critério Análise de Mutantes. A análise do *strength* pode fornecer informações importantes na determinação de estratégias de teste com baixo custo e alta probabilidade de encontrar defeitos.

Espera-se que os artefatos gerados neste trabalho permitam a repetição do experimento e a verificação da evolução do conhecimento pertinente, contribuindo para a evolução e desenvolvimento de ferramentas que auxiliem a atividade de teste. Devido às características dos artefatos gerados, espera-se que este também possa ser aproveitado como material de ensino e treinamento das técnicas de teste avaliadas (Funcional, Estrutural e Teste de Mutação). Como o material gerado é baseado em programas utilizados para o ensino de algoritmos de programação, a introdução dos conceitos e técnicas de teste de software pode ser feita juntamente com o ensino da programação e assim reforçar a importância de testar os programas durante o desenvolvimento dos mesmos.

1.3 Organização

Neste capítulo foram apresentados o contexto, as motivações e os objetivos deste trabalho. No Capítulo 2 são introduzidos os principais conceitos de teste de software e Engenharia de Software Experimental. São descritas algumas ferramentas que apoiam os critérios de testes e algumas características do paradigma OO que podem causar algum impacto durante a atividade de testes. No Capítulo 3 são apresentadas as definições e o planejamento do estudo. No Capítulo 4 são descritas todas as atividades executadas, as dificuldades encontradas e as soluções aplicadas. No Capítulo 5 são apresentados os resultados da aplicação dos testes nos dois paradigmas e a avaliação das hipóteses definidas. Por fim as conclusões e contribuições deste trabalho são apresentados no Capítulo 6.

CAPÍTULO

2

Fundamentos de Teste de Software e Engenharia de Software Experimental

2.1 Considerações Iniciais

A atividade de teste de software, muitas vezes subestimada durante o desenvolvimento de sistemas, vem sendo a cada dia mais valorizada por empresas preocupadas com a qualidade do software produzido (Myers *et al.*, 2004).

Dada a diversidade de critérios que têm sido estabelecidos, um ponto crucial que se coloca nessa perspectiva é a escolha e/ou a determinação de uma estratégia de teste, que em última análise passa pela escolha de critérios de teste, de forma que as vantagens de cada um desses critérios sejam combinadas objetivando uma atividade de teste de maior qualidade. Estudos teóricos e experimentais de critérios de teste são de extrema relevância para a formação desse conhecimento, fornecendo subsídios para o estabelecimento de estratégias de baixo custo e alta eficácia (Maldonado *et al.*, 1998).

Prado (2009) observou que estudos que consideram os diferentes critérios de teste em diferentes paradigmas de programação são raros. Com isso, foi iniciado um estudo com o objetivo de avaliar o impacto do paradigma de programação na atividade de teste. Em seu trabalho foram avaliadas as técnicas Funcional e Estrutural sob a perspectiva procedimental e OO. O

estudo apresentado nesta dissertação de mestrado busca complementar o trabalho de Prado e avaliar o critério Análise de Mutantes nos dois paradigmas de programação.

Para a compreensão deste estudo, é necessário o conhecimento de alguns conceitos relacionados ao teste de software e das principais técnicas e critérios existentes. Também é importante conhecer os conceitos básicos da engenharia de software experimental que oferece um modo sistemático, disciplinado, computável e controlado para avaliação dos critérios de teste (Travassos, 2002).

Este capítulo está organizado da seguinte forma. Na Seção 2.2, os principais conceitos e terminologias sobre teste de software são apresentados. As fases da atividade de testes são descritas de uma forma genérica, independentemente do paradigma de programação. As técnicas de teste são descritas na Seção 2.3. Naquela seção também são apresentados os principais critérios de testes e os requisitos e condições de satisfação de cada um. Na Seção 2.4, é discutida a necessidade de automatização dos testes, bem como iniciativas para o desenvolvimento de ferramentas que auxiliem esta atividade. Na Seção 2.5, são mostrados os principais conceitos adicionados ao teste de software no paradigma OO em relação ao paradigma Procedimental e como esses conceitos podem afetar o teste de software. Os fundamentos da Engenharia de Software Experimental são apresentados na Seção 2.6. Naquela seção também são discutidos os diferentes tipos de experimentos, as características de cada um e as principais fases do processo de experimentação. Por último, na Seção 2.7, são apresentados alguns estudos teóricos e experimentais avaliando diferentes técnicas e critérios de teste em diferentes perspectivas.

2.2 Terminologia e Conceitos Básicos de Teste de Software

Os conceitos apresentados nesta seção foram descritos com base em Maldonado *et al.* (1998) e Delamaro *et al.* (2007b).

As atividades de garantia de qualidade de software são divididas em validação, verificação e testes. A verificação e validação são consideradas atividades estáticas que visam a assegurar que o software cumpra com suas especificações e atenda as necessidades do usuário. A verificação procura identificar se o produto está sendo construído corretamente, seguindo o processo de desenvolvimento estabelecido e gerando as saídas desejadas. A validação diz respeito à conformidade do produto construído em relação ao produto almejado pelo cliente.

Segundo Delamaro *et al.* (2007b), o teste de software é uma atividade dinâmica cujo intuito é executar o programa ou modelo utilizando algumas entradas em particular e verificar se seu comportamento está de acordo com o esperado. O domínio de entrada de um programa é o

conjunto de todos os valores aceitos na descrição do programa. O domínio de saída é definido pelo conjunto de todos os valores gerados pelo programa. Um "dado de teste" para um programa P é um elemento do domínio de entrada de P. Um "caso de teste" é um par formado por um dado de teste mais o resultado esperado para a execução do programa com aquele dado de teste. Ao conjunto de todos os casos de testes usados durante uma determinada atividade de teste costuma-se chamar "conjunto de teste" ou "conjunto de casos de teste".

O objetivo da atividade de teste, ao contrário do que pode parecer, não é garantir que o programa não tenha erros mas sim tentar encontrar situações em que o programa apresente as falhas que existem (Myers *et al.*, 2004). Com isso, pode-se concluir que um teste de sucesso é aquele que é capaz de encontrar um defeito ainda não revelado.

É importante conhecer a nomenclatura utilizada na literatura para distinguir os termos defeito, engano, erro e falha:

- Defeito (*fault*) - Passo, processo ou definição de dados incorretos. É considerado um defeito um comando ou instrução que foi escrito de forma incorreta;
- Engano (*mistake*) - Ação humana que produz um defeito, como por exemplo uma ação incorreta tomada pelo programador;
- Erro (*error*) - Diferença entre o valor obtido e o valor esperado durante a execução do programa, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro;
- Falha (*failure*) - Produção de uma saída incorreta do programa quando comparada com a saída esperada na especificação.

Com base nessas definições, um programador pode cometer um engano durante a criação do programa e inserir um defeito no código-fonte. Caso a instrução com defeito seja executada ela pode fazer com que o programa alcance um estado incorreto, caracterizando assim um erro. Se esse erro interferir no resultado produzido pelo programa pode-se dizer que uma falha ocorreu. Delamaro *et al.* (2007b) ressaltam que essas definições não são seguidas o tempo todo e nem são unanimidade entre os pesquisadores e engenheiros de software.

2.2.1 Fases de Teste

Para que o processo de teste de software possa ser melhor gerenciado, recomenda-se sua aplicação em etapas ou fases. Assim, essa atividade pode ser aplicada em três fases: teste de unidade, teste de integração e teste de sistema.

- **Teste de unidade:** concentra os testes na menor unidade do projeto de software. O objetivo nessa fase é de encontrar erros de lógica e de implementação em cada módulo do software separadamente. Durante os testes de unidade muitas vezes métodos, procedimentos ou funções precisam ser invocados. Quando esses componentes não estão completamente implementados faz-se necessário o uso de *drivers* e *stubs*. *Driver* é uma unidade responsável por executar a unidade testada, invocando-a com os parâmetros adequados e coletando seus resultados. *Stubs* são unidades que simulam o comportamento de uma unidade na qual a unidade testada precisa invocar. A menor unidade de um programa pode variar de acordo com o paradigma, maiores detalhes sobre essa discussão serão apresentados na Sessão 2.5.

- **Teste de integração:** busca identificar erros existentes na interface entre as unidades. O teste de integração exercita a interação entre as unidades e pode ser executado seguindo duas estratégias: Na integração *top-down* os componentes de mais alto-nível são testados primeiro e então os testes são executados sobre os componentes de níveis inferiores. Na abordagem *bottom-up* os componentes de níveis mais baixo são testados e à medida que os níveis inferiores são validados os componentes que os invocam são testados.

- **Teste de sistema:** após a integração e teste de todos os componentes é necessário verificar sua compatibilidade com recursos externos. O teste de sistema é responsável por garantir que o produto desenvolvido seja compatível com esses recursos, tais como: banco de dados, sistema operacional, hardware, etc. Nessa etapa de teste os requisitos não funcionais também são verificados, como por exemplo, requisitos de desempenho, estresse e segurança.

Independentemente da fase de teste, exitem algumas etapas bem definidas para a execução da atividade de teste (Delamaro *et al.*, 2007b). Em Sommerville (2003) são descritas as quatro etapas principais da atividade de teste:

- Planejamento de Testes: fase em que são definidos como os testes serão aplicados. De acordo com os recursos disponíveis, são planejados o cronograma e as responsabilidades de cada integrante durante os testes. Nesta fase são definidos também os objetivos, os métodos e ferramentas adotadas para a realização dos testes (Pressman, 2005).;

- Projeto dos Casos de Teste: nessa fase é projetada uma estratégia para a construção dos casos de testes com o objetivo de cobrir as condições de satisfação dos critérios estabelecidos;

- Execução do Teste: com o auxílio ou não de uma ferramenta, os casos de testes construídos na fase anterior são executados no programa em teste e sua saída comparada com a saída esperada, caso algum erro seja encontrado, estes são documentados para que as causas sejam identificadas e posteriormente corrigidas; e
- Coleta e avaliação dos resultados dos testes: nessa etapa, os resultados dos testes realizados são registrados, organizados e apresentados na forma de relatórios.

2.3 Técnicas de Teste

Além dos conceitos básicos de teste de software é importante conhecer as principais técnicas e critérios existentes. Os critérios de testes servem para guiar o testador na construção casos de testes e também podem fornecer uma medida de qualidade dos conjuntos de CT construídos de acordo com os requisitos definidos por cada critério (DeMillo, 1980). De acordo com Delamaro *et al.* (2007b) um conjunto de teste que satisfaz todos os requisitos de um critério de teste C é dito "adequado" a C, ou "C-adequado". A seguir são apresentados as principais técnicas e critérios de testes existentes:

2.3.1 Teste Funcional

Os critérios baseados na técnica Funcional são derivados da especificação do software. Como os aspectos de implementação não são considerados, essa técnica também é denominada teste caixa-preta. Com base nos conceitos descritos em Fabbri *et al.* (2007) os principais critérios pertencentes a técnica Funcional descritos a seguir:

Particionamento em Classes de Equivalência: a partir das funcionalidades descritas no documento de especificação, os possíveis valores do domínio de entrada são divididos em classes representativas. As classes são separadas em válidas ou inválidas dependendo da aceitação ou não dos dados pelo programa testado. Esse critério se baseia na hipótese de que um elemento de uma determinada classe representa o comportamento de qualquer outro elemento contido na mesma classe.

A condição de satisfação para o critério particionamento em classes de equivalência é que o conjunto de casos de teste cubra todas as classes de equivalência com pelo menos um caso de teste para cada classe inválida. Esse critério pode ser estendido para que as classes de equivalência sejam consideradas também para as saídas do programa.

Análise de Valor Limite: esse critério é complementar ao particionamento em classe de equivalência. Além de exigir que todas as classes de equivalência sejam cobertas pelo conjunto

de casos de teste, o critério exige que os limites de cada classe sejam exercitados. De acordo com Myers *et al.* (2004), a experiência mostra que casos de testes que exploram condições limites tem uma maior probabilidade de encontrar defeitos. Tais condições correspondem a valores que estão exatamente sobre ou imediatamente acima ou abaixo dos limitantes das classes de equivalência (Fabbri *et al.*, 2007).

Teste Funcional Sistemático (*Systematic Functional Testing*): é uma variante mais forte do critério particionamento em classes de equivalência, seguindo um conjunto bem definido de diretrizes (Linkman *et al.*, 2003). Após a partição das classes do domínio de entrada, o Teste Funcional Sistemático exige no mínimo dois casos de testes para cada partição. O critério possui uma série de diretrizes para construção dos casos de testes relacionados a casos especiais, valores ilegais, vetores, valores reais entre outros.

2.3.2 Teste Estrutural

Na técnica de teste estrutural, também conhecida como teste caixa branca (em oposição ao nome caixa preta), os aspectos de implementação são fundamentais na escolha dos casos de teste (Maldonado *et al.*, 1998). Os conceitos apresentados nesta sub-seção são baseados na síntese descrita em Barbosa *et al.* (2007).

Em geral, a técnica Estrutural utiliza como auxílio na definição dos requisitos uma representação do código-fonte denominada Grafo de Fluxo de Controle (GFC). Para representar o programa em um GFC é preciso decompor o mesmo em blocos. Um bloco de código é um trecho do programa em que se o primeiro comando é executado todos os outros comandos do bloco também serão. Dessa forma todos os comandos contidos em um bloco, com exceção do primeiro e do último, possuem exatamente um comando predecessor e um sucessor. Os blocos são representados como nós no GFC e a correspondência entre os blocos indicando possíveis fluxos de controle são representada por arcos. De acordo com esses conceitos um GFC pode ser definido como um grafo orientado com apenas um nó de entrada podendo ter diversos nós de saída correspondendo ao inicio e ao fim do programa respectivamente, sendo que cada nó corresponde a um bloco indivisível de comandos e cada aresta um possível desvio para algum bloco.

Seja um grafo de fluxo de controle $G = (N, E, s)$ onde N representa o conjunto de nós, E o conjunto de arcos, e s o nó de entrada. Um caminho é uma sequência finita de nós (n_1,n_2, ... ,n_k), $k \geq 2$, tal que existe um arco de n_i para $n_(i + 1)$ para i = 1,2, ... ,k-1.

A principal dificuldade em aplicar o teste Estrutural encontra-se na determinação dos "caminhos não executáveis". Um caminho não executável é um caminho do GFC que não pode ser

coberto por nenhum valor do domínio de entrada. A identificação de caminhos não executáveis é um problema muitas vezes indecidível.

Com base no GFC diversos critérios são propostos, diferenciando-se em relação aos elementos exigidos para determinar seus requisitos. Os critérios da técnica Estrutural são, basicamente, divididos em Fluxo de Controle, Fluxo de Dados e em Complexidade.

Critérios Baseados em Fluxo de Controle

Utilizam apenas informações referentes ao fluxo de controle, como blocos de comandos e desvios. Os principais critérios deste grupo são:

Todos-Nós: esse critério exige que os casos de testes executem ao menos uma vez cada vértice do GFC, ou seja, exige que cada comando do programa seja executado pelo menos uma vez.

Todos-Arcos: também conhecido como Todas-Arestas, requer que cada desvio do fluxo de controle do programa seja exercitado pelo menos uma vez, cobrindo assim todas as arestas do grafo.

Todos-Caminhos: o mais completo dentre os critérios citados. Exige que todos os caminhos possíveis do GFC sejam executados. Apesar de ser um critério bastante forte, na maioria das vezes ele é reconhecido como impraticável, devido à quantidade de caminhos possíveis.

Critérios Baseados em Fluxo de Dados

Os requisitos de satisfação desses critérios são baseados em informações referentes ao fluxo de dados do programa. O foco deste grupo de critérios é na interação entre definições e usos de variáveis. Dentre os critérios de fluxo de dados, destacam-se os propostos por Rapps e Weyuker (1985) que utilizam uma extensão do GFC denominada Grafo Def-Uso.

Nesse grafo, além das arestas e nós criados no GFC, são inseridas também informações a respeito do fluxo de dados. Dessa forma é possível identificar pontos em que um valor é atribuído a uma variável (caracterizando uma definição de variável) e pontos onde esses valores atribuídos são utilizados (caracterizando um uso de variável). O uso de uma variável pode ocorrer de duas formas: caso seja utilizado para determinar o desvio do fluxo de controle do programa ele é chamado de uso predicativo (p-uso). Quando o valor da variável é utilizado em uma computação ele é classificado como uso computacional (c-uso).

A maioria dos critérios que utilizam essa extensão do GFC exigem que os caminhos entre os elementos requeridos sejam caminhos livres de definição, ou seja, um caminho no qual a

variável avaliada não é redefinida. Alguns critérios definidos a partir do Grafo Def-Uso são (Rapps e Weyuker, 1985):

Todas-Definições: esse critério exige que toda definição de variável seja exercitada uma vez, seja por meio de um c-uso ou por um p-uso. O critério é baseado na hipótese de que se um valor é atribuído a alguma variável, então essa variável deve ser utilizada posteriormente, caso contrário, não existe a necessidade de tal atribuição.

Todos-Usos: requer que todas as associações entre uma definição de variável e seus subsequentes usos sejam exercitadas pelos casos de testes por pelo menos um caminho livre de definição, ou seja, um caminho em que a variável não redefinida. Os critérios Todos-p-Usos, Todos-p-Usos/Alguns-c-Usos e Todos-c-Usos/Alguns-p-Usos representam variações do critério Todos-Usos.

Todos-Du-Caminhos: requer que toda associação entre uma definição de variável e subsequentes p-usos ou c-usos dessa variável seja exercitada por todos os caminhos livres de definição e livres de laço que cubram essa associação.

A principal desvantagem dos critérios definidos por Rapps e Weyuker (1985) é que seus requisitos são baseados na ocorrência explicita de um uso de variável, e que na presença de caminhos não executáveis, estes critérios não garantem que todos os arcos sejam cobertos. Nesse contexto, Maldonado (1991) definiu a família de critérios **Potenciais-Usos**.

Em linhas gerais, os critérios Potenciais-Usos requerem associações independentemente da ocorrência explícita de uma referência (um uso) a uma definição de variável. Ou seja, se um uso dessa definição pode existir (há um caminho livre de definição até um certo nó ou arco, isto é, um potencial-uso), a potencial-associação entre a definição e o potencial-uso é caracterizada e eventualmente requerida (Barbosa *et al.*, 2007).

Todos-Potenciais-Usos é o principal critério dessa família. Ele é satisfeito quando são construídos casos de testes que exercitem pelo menos um caminho livre de definição de uma variável definida em um nó i para todo nó e todo arco possível de ser alcançado a partir daquela definição (Maldonado, 1991).

2.3.3 Teste de Mutação

Com base nos conceitos descritos em Delamaro *et al.* (2007a) são apresentados a seguir as principais características do teste de mutação também conhecido como critério Análise de Mutantes.

No critério **Análise de Mutantes** (DeMillo *et al.*, 1978) pequenas modificações sintáticas são realizadas sobre o código-fonte. Esses programas ligeiramente alterados são denominados

mutantes e o objetivo do critério é gerar casos de testes que, a partir da mesma entrada, os resultados gerados pelo programa original seja diferente dos resultados gerados pelo programa mutante.

Para modelar as modificações nos programas mutantes, são utilizados "operadores de mutação". Um operador de mutação aplicado a um programa P transforma P em um programa similar, ou seja, um mutante (Delamaro *et al.*, 2007a). Além de modelar alguns tipos de defeitos cometidos por programadores, os operadores de mutação podem ser definidos para exigir algumas características específicas dos casos de testes, como por exemplo a cobertura de todos os comandos dos programas. Delamaro *et al.* (2007a) define esses mutantes como "mutantes instrumentados". Como cada operador de mutação pode ser aplicado a diversas partes do código testado, a quantidade de mutantes gerados pode ser muito grande, o que pode tornar a aplicação da técnica custosa.

Após a geração dos mutantes, casos de testes devem ser construídos com o objetivo de evidenciar as diferenças entre o programa mutante e o programa original. Uma das formas de identificar as diferenças entre os dois programas é avaliar o resultado gerado por eles na saída padrão. Se um mutante M gerar uma saída diferente do programa original para um determinado caso de teste, pode-se dizer que M está morto. Caso o mutante M apresente sempre o mesmo resultado que o programa original para qualquer valor de entrada contido no domínio de entrada do programa testado, M é classificado como mutante equivalente, sendo esse impossível de ser morto pelos casos de testes.

A primeira etapa do teste de mutação é a geração dos mutantes a partir dos operadores de mutação selecionados. Após esta etapa, um conjunto inicial de casos de teste deve ser executado contra o programa original e caso as saídas geradas estejam de acordo com as saídas esperadas, esse conjunto de casos de teste é executado contra os programas mutantes. Após a execução dos programas mutantes com os casos de teste e identificação dos mutantes mortos, os mutantes sobreviventes devem ser analisados. Durante essa análise duas situações são possíveis: caso seja identificado um mutante equivalente, o mesmo deve ser marcado e o escore de mutação atualizado. Caso contrário novos casos de teste devem ser construídos para evidenciar as diferenças entre o mutante e o programa original.

Uma outra característica que encarece a aplicação do critério Análise de Mutantes é a na identificação de mutantes equivalentes. Um mutante equivalente é um programa mutante que gera as mesmas saídas que o programa original para qualquer valor pertencente ao domínio de entrada aceito pelo programa. A determinação de programas equivalentes é indecidível e precisa ser feita manualmente pelo testador.

A medida de cobertura é uma importante forma de avaliar a qualidade do conjunto de casos de teste construído. Ela fornece informações que permitem que o testador decida se um conjunto

de casos de testes é suficiente ou não para determinado programa. A medida de cobertura do critério Análise de Mutantes é o escore de mutação que é calculado como a razão entre o número de mutantes mortos e o número de mutantes não equivalentes. O escore de mutação varia entre 0 e 1 e fornece uma medida objetiva de quanto o conjunto de casos de teste analisado aproxima-se da adequação (Delamaro *et al.*, 2007a).

2.4 Ferramentas de Apoio ao Teste de Software

A atividade de teste é reconhecidamente uma atividade de alto custo e geralmente devido a falta de recursos disponíveis ela é realizada somente nas últimas fases do desenvolvimento de software (Myers *et al.*, 2004). Ferramentas que auxiliem os testes são importantes pois além de facilitar a aplicação correta da técnica, ajudam o testador na construção e execução dos casos de testes. A automatização de algumas atividades contribui também para a redução dos custos e diminui a possibilidade do testador cometer enganos.

Apesar dos benefícios evidentes, poucas ferramentas foram desenvolvidas com o objetivo de cobrir todas as fases de testes. As ferramentas existentes, em geral, focam na aplicação de uma técnica específica para uma determinada linguagem de programação.

A técnica **Funcional** não exige uma rigorosidade tão grande quando comparado com outras técnicas, e por isso poucas ferramentas são construídas para dar suporte a seus critérios. O particionamento em classe de equivalência e a análise de valor limite geralmente são documentadas utilizando apenas um editor de texto e testadas a partir da execução do programa.

O *framework* JUnit (Riehle, 2008), apesar de não se basear em nenhuma técnica, pode auxiliar a atividade de testes pois, permite a organização e execução dos casos de teste. O JUnit permite também a automatização do teste de regressão. O teste de regressão é caracterizado pela re-execução de todos os casos de teste quando necessário. Essa atividade ocorre geralmente durante a fase de correção dos erros encontrados. Assim, após o programa ser corrigido pelos desenvolvedores os testes precisam ser executados novamente para garantir que os erros foram corrigidos e novos erros não foram introduzidos.

Da mesma forma que o JUnit auxilia a atividade de teste para programas em Java, a ferramenta CUTE (Sommerlad e Graf, 2007) oferece suporte para programas desenvolvidos nas linguagens C e C++.

Como os requisitos da técnica **Estrutural** são gerados a partir do código-fonte e aqueles exigem um considerável esforço para serem identificados, diversas ferramentas foram desenvolvidas com o objetivo de automatizar algumas atividades dos critérios desta técnica. Em

Barbosa *et al.* (2007); Maldonado *et al.* (1998) são descritas diversas ferramentas que apoiam a técnica Estrutural, dentre elas destacam-se:

- **ASSET** (Frankl e Weyuker, 1985): ferramenta desenvolvida na *New York University* em 1985 para o teste de programas Pascal com suporte aos critérios baseados na análise de fluxo de dados.

- **xSuds** (Agrawal *et al.*, 1998): ambiente desenvolvido com o objetivo de fornecer uma plataforma de testes contendo um conjunto de ferramentas de apoio a geração de casos de teste, a análise e depuração de programas. Atualmente o ambiente xSuds vem sendo comercializado pela IBM com suporte aos critérios de teste Estrutural em programas escritos em C e C++. Para o teste de programas em C a ferramenta apoia a aplicação dos critérios de fluxo de controle (Todos-Nós e Todos-Arcos) e de fluxo de dados (Todos-c-Usos, Todos-p-Usos e Todos-Usos) e para a linguagem C++ apenas os critérios baseados no fluxo de controle.

- **Poke-tool** (Chaim, 1991): ferramenta proposta inicialmente para suporte a testes aplicados as linguagens C, devido à sua característica, também foram propostas configurações para o teste de programas em Fortran e Cobol (Fonseca, 1993; Leitão, 1992). A ferramenta oferece auxilio aos critérios, Todos-Nós, Todos-Arcos, Todos-Usos e Todos-Potenciais-Usos. A ferramenta é orientada à sessão e algumas de suas funcionalidades incluem: geração de sessão de teste; inserção e execução de casos de teste; instrumentação do programa sob teste; avaliação da cobertura dos testes, etc.

- **JaBUTi** (Vincenzi *et al.*, 2003): ferramenta desenvolvida para apoiar a aplicação dos critérios da técnica Estrutural para a linguagem Java. Ela suporta os principais critérios desta técnica, como Todos-Nós, Todos-Arcos, Todos-Usos e Todos-Potenciais-Usos, incluindo os critérios dependentes e independentes de exceção.

  Uma característica importante dessa ferramenta é que ela permite que os testes sejam realizados com base no *bytecode*. Isso é importante no teste de componentes, pois nem sempre o código-fonte está disponível para o testador (Vincenzi *et al.*, 2007). Dentre as suas funcionalidades podem ser destacadas a possibilidade de importação dos casos de teste do JUnit, a visualização gráfica do GFC, o acompanhamento dos critérios já cobertos e os requisitos necessários para a satisfação dos critérios.

  Atualmente a JaBUTi se caracteriza por uma família de ferramentas que estão sendo desenvolvidas para dar suporte a diversas áreas, como a Orientação a Aspectos (OA) (Lemos, 2009) e a Arquitetura Orientada a Serviços (SOA) (Eler, 2008).

Também foram construídas diversas ferramentas para apoiar o critério **Análise de Mutantes**. Tradicionalmente, não existe uma maneira direta de definir os operadores de mutação para uma dada linguagem. Em geral, os operadores de mutação são projetados tendo por base a experiência no uso de dada linguagem, bem como os enganos mais comuns cometidos durante sua utilização (Delamaro *et al.*, 2007a). As principais ferramentas que suportam este critério são:

- **Mothra** (Choi *et al.*, 1989): segundo Delamaro *et al.* (2007a), foi a ferramenta que mais se popularizou e mais influenciou o desenvolvimento do critério. A ferramenta foi desenvolvida por pesquisadores da Purdue University e do Georgia Institute of Technology para programas em Fortran. Na ferramenta é possível selecionar o conjunto de operadores de mutação a serem aplicados e a "intensidade" do teste, que é a porcentagem dos mutantes que deve ser aleatoriamente, selecionada.

- **Proteum** (Delamaro, 1993): de acordo com Delamaro *et al.* (2007a), foi desenvolvida para dar suporte ao critério Análise de Mutantes para programas desenvolvidos em C. A versão atual da ferramenta foi quebrada em diversos programas, cada um com uma funcionalidade especifica. Cada um desses programa pode ser executados a partir da linha de comandos do console e também dentro de *scripts*.

  A ferramenta disponibiliza relatórios estatísticos, mostrando a quantidade de mutantes que cada caso de teste matou, quantos ainda estão vivos, quantos foram marcados como equivalentes, etc. A ferramenta conta com 71 operadores de mutação divididos em quatro classes: mutação de comandos, mutação de operadores, mutação de variáveis e mutação de constantes.

  Atualmente existem diversas linhas de pesquisa em torno dessa ferramenta, dentre elas destacam-se a extensão da ferramenta para aplicação em Redes de Petri (Simão, 2000), Máquinas de Estados Finitos (Fabbri *et al.*, 1995) e Statecharts (Sugeta *et al.*, 1999).

- **Jumble** (Irvine *et al.*, 2007): ferramenta de apoio ao critério para a linguagem Java. Pode ser utilizada por linha de comando ou como um plugin do Eclipse. Quando utilizada no Eclipse a ferramenta trabalha em conjunto com o jUnit para a construção e execução dos casos de teste. Não permite a seleção dos operadores de mutação e suas saídas são todas em forma textual.

- **JesTer** (Moore, 2008) : também desenvolvida para a linguagem Java com integração com casos de teste utilizando o JUnit. Os resultados da execução dos mutantes são salvos em um arquivo XML o que facilita a comunicação com outras ferramentas.

- **MuJava** (Offutt *et al.*, 2006): é o resultado de uma colaboração entre duas universidades, Korea Advanced Institute of Science e Technology (KAIST) e George Mason University. A ferramenta trabalha com 43 operadores, sendo 15 operadores do nível de método e 28 operadores do nível de classe, permitindo ao testador selecionar quais operadores deverão ser considerados.

- **MuClipse** (Smith e Williams, 2009): construída a partir da MuJava, foi exportada como um *plugin* para o ambiente Eclipse com o objetivo de simplificar a instalação e configuração da ferramenta. Também é integrada com o JUnit e utiliza o mesmo conjunto de operadores da MuJava. Os resultados da execução dos casos de teste são mostrados no terminal do eclipse indicando quantos mutantes foram mortos, quantos ainda estão vivos e o escore de mutação obtido. A ferramenta permite também visualizar quais são os mutantes vivos e qual a sua diferença em relação ao programa original.

As ferramentas apresentadas nesta seção foram desenvolvidas para dar suporte a diversas linguagens de programação sobre diferentes paradigmas. Na próxima seção são apresentados algumas diferenças entre os paradigmas Procedimental e Orientado a Objetos e como essas diferenças podem influenciar na atividade de teste.

2.5 Impacto do Paradigma de Programação na Testabilidade

As técnicas e critérios apresentadas na Seção 2.3 foram, em geral, definidas com base no paradigma procedimental, também denominado paradigma tradicional ou técnica estruturada. O paradigma Orientado a Objetos altera a organização da estrutura do código-fonte o que pode alterar também a forma de executar os testes. Os conceitos apresentados nesta seção foram parcialmente extraídos de Vincenzi *et al.* (2007).

Mudanças no paradigma da linguagem de implementação podem afetar a atividade de testes de diversas maneiras, como por exemplo, dificultando o acesso a métodos ou criando novas possibilidades para o fluxo de controle. O conjunto de instruções definido para uma linguagem de programação também afeta significativamente a atividade de teste, principalmente para técnicas nas quais os critérios são definidos de acordo com as instruções da linguagem, como é o caso do critério Análise de Mutantes.

2.5.1 Conceitos do Paradigma OO

O paradigma OO muitas vezes é visto como uma evolução do paradigma Procedimental. De acordo com Vincenzi *et al.* (2007), a idéia por trás da orientação a objetos é agrupar em uma entidade (denominada classe) os dados (atributos) e os procedimentos/funções (métodos) que realizam as operações sobre os dados. O uso da orientação a objetos não é capaz de garantir o correto funcionamento de um programa por si próprio. Pelo contrário, o que se tem observado é que as características principais da orientação a objetos introduzem novas fontes de falhas de modo que atividades de VV&T são de fundamental importância no contexto de programas orientados a objetos.

A seguir são discutidos alguns problemas que podem ocorrer em programas OO devido aos conceitos existentes neste paradigma.

Encapsulamento: É o mecanismo responsável por controlar o acesso de entidades externas aos atributos e métodos contidos na classe. Dessa forma é possível tornar visível à outras classes apenas a interface, ocultando detalhes da implementação. Se usado corretamente o encapsulamento favorece o ocultamento de informação e a modularização do sistema.

O encapsulamento pode dificultar a realização dos testes, prejudicando a controlabilidade e a observabilidade da entidade testada. Atributos e métodos invisíveis a entidades externas dificultam a identificação do estado do objeto e até mesmo o teste de métodos que tenham seu acesso restrito. A linguagem C++ fornece um meio de contornar esse problema com a utilização de funções amigas (*friend functions*). Harrold (2000) e Rosa e Martins (1988) sugerem outras abordagens para contornar esse obstáculo em linguagens que não oferecem o mesmo recurso de C++.

Herança: a herança permite a reusabilidade via o compartilhando de características já definidas em outra classes. Entretanto, como destacado por Binder (1996), a herança enfraquece o encapsulamento e pode ser responsável pela criação de um risco de defeito similar ao uso de variáveis globais em programas procedimentais. Outro problema associado a esse conceito é que se usado excessivamente pode dificultar a compreensão, aumentando as chances do programador cometer enganos.

Polimorfismo: é caracterizado pela possibilidade de um único nome (denominado nome polimórfico), ou seja, uma variável ou uma referência, denotar instâncias (objetos) de várias classes que, usualmente devem possuir uma única superclasse em comum (Vincenzi *et al.*, 2007).

Segundo Vincenzi *et al.* (2007), um dos problemas causados pelo polimorfismo é a indecidibilidade no acoplamento dinâmico. Uma vez que nomes polimórficos podem denotar objetos

de diferentes classes, é impossível, ao invocar um método por meio de um nome polimórfico, predizer em tempo de compilação qual o trecho de código será executado, ou seja, se será executado o método original da superclasse ou o método refinado de uma possível subclasse. Tal decisão só ocorre em tempo de execução

2.5.2 Impacto nas Fases e Critérios de Teste

Além do impacto causado diretamente pelos conceitos definidos para o paradigma OO, algumas definições podem ser interpretadas de maneira diferente quando o paradigma é alterado. Durante a divisão das atividade de testes em fases por exemplo, não existe um consenso em relação a definição da menor unidade a ser testada durante a fase de teste unitário no paradigma OO.

Segundo o padrão IEEE (1990), uma unidade é um componente de software que não pode ser subdividido. Com base nessa informação fica claro que para o paradigma procedimental a menor unidade a ser testada é a função ou procedimento. Essa classificação não é tão simples quando o paradigma OO é estudado.

Vincenzi. (2004) defende que a menor unidade em programas OO são os métodos. Nesse sentido, a classe é vista como o *Driver* do método, pois sem ela o método não pode ser executado. Assim, durante a fase de teste de unidade os testes utilizando o paradigma tradicional se resumem a testes intraprocedimentais, enquanto que no paradigma OO são realizados testes intramétodos.

Para o teste de integração no paradigma Procedimental, são exercitadas as interações entre os procedimentos e funções, por isso, esse tipo de teste é denominado teste interprocedimental. Métodos de uma mesma classe podem interagir entre si trabalhando em conjunto para compor determinada funcionalidade. O teste baseado nessa interação é chamado de teste intermétodo (Harrold e Rothermel, 1994). Entretanto, este teste não cobre todas as interações das unidades do paradigma OO, por isso em Harrold e Rothermel (1994) são propostos mais dois tipos de testes: teste intraclasse e teste interclasse. O teste intraclasse tem foco nas interações entre os métodos públicos de uma mesma classe, com o objetivo de identificar sequências que levam o objeto a estados inconsistentes. O mesmo acontece com o teste interclasse, porém, ele abrange uma quantidade maior de possibilidades, pois não exige que os métodos públicos sejam da mesma classe.

Na Figura 2.1, adaptada de (Binder, 1999) por (Maldonado *et al.*, 2005) as principais fases do teste de software e as diferenças na classificação entre o paradigma procedimental e orientado a objetos podem ser identificadas. Do lado esquerdo as fases de testes podem ser vistas em relação ao paradigma procedimental, durante o teste de unidade as funções são exercitadas

individualmente, seguindo para o teste de integração para então testar o sistema como um todo. Do lado direito as fases são apresentadas de acordo com as definições de Vincenzi. (2004).

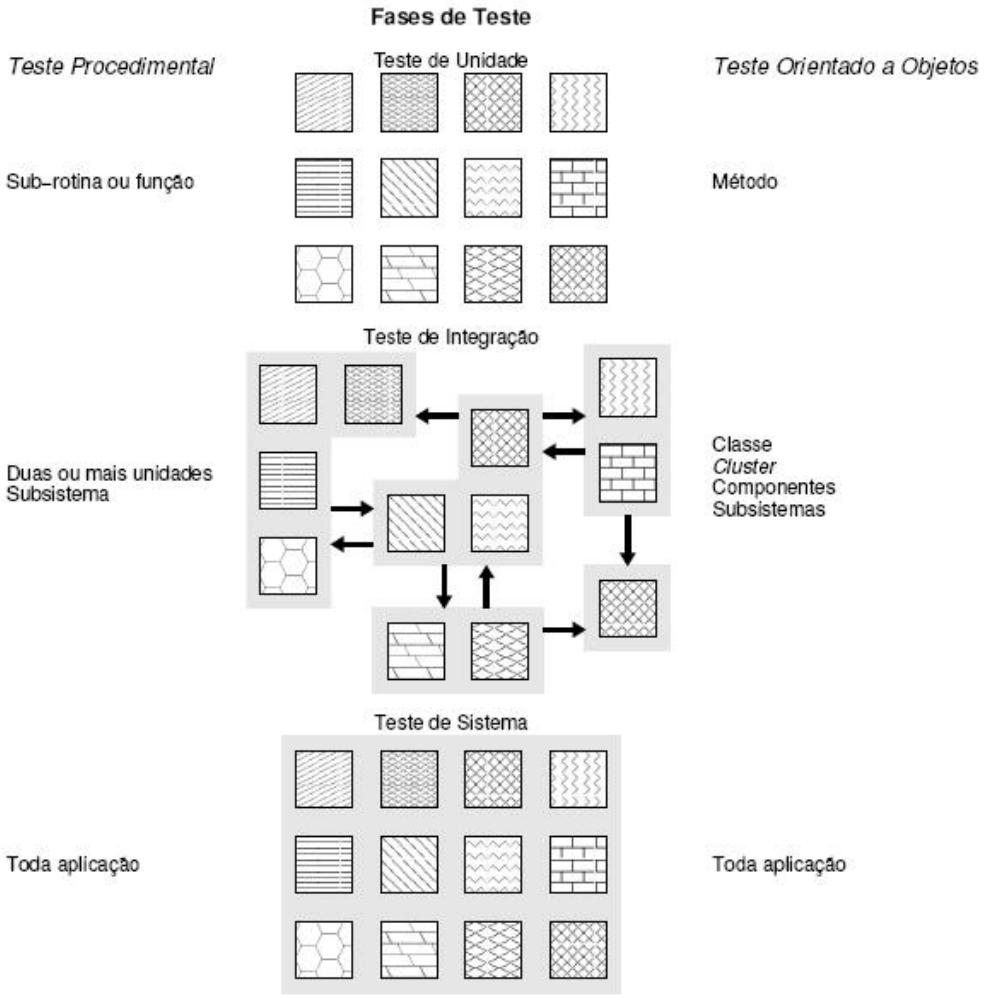

Figura 2.1: Relacionamento entre teste de unidade, de integração e de sistema: programas procedimentais e OO extraídos de (Maldonado *et al.*, 2005)

Alguns autores entendem que a menor unidade de um programa OO é a classe e não o método (Binder, 1999; Rountev *et al.*, 2004). Dessa forma os testes intramétodo, intermétodo e intraclasse devem ser executados durante a fase de teste de unidade, enquanto no teste de integração apenas o teste interclasse deve ser executado. Na Tabela 2.1 é apresentada uma síntese dos tipos de testes que podem ser aplicados em cada uma das fases tanto em programas procedimentais quanto em programas OO considerando o método e a classe como a menor unidade.

Tabela 2.1: Relação entre fases de teste de programas procedimentais e OO (Vincenzi., 2004)

Menor Unidade: Método		
Fase	**Teste Procedimental**	**Teste Orientado a Objetos**
Unidade	Intraprocedimental	IntraMétodo
Integração	Interprocedimental	InterMétodo, IntraClasse e InterClasse
Sistema	Toda Aplicação	Toda Aplicação
Menor Unidade: Classe		
Fase	**Teste Procedimental**	**Teste Orientado a Objetos**
Unidade	Intraprocedimental	IntraMétodo, InterMétodo e IntraClasse
Integração	Interprocedimental	InterClasse
Sistema	Toda Aplicação	Toda Aplicação

Além da discussão em relação a menor unidade a ser testada no teste de unidade, o paradigma de programação pode influenciar diretamente nos requisitos definidos pelas técnicas e critérios existentes. Os critérios baseados na técnica Funcional são os que possuem menor dependência em relação ao paradigma de implementação, uma vez que os requisitos gerados por estes critérios são baseados apenas na especificação do programa.

Os critérios baseados na técnica Estrutural podem sofrer uma maior influência do paradigma de programação, pois seus requisitos são construídos a partir do Grafo de Fluxo de Controle e do Grafo-Def-Uso, que por sua vez são derivados do código-fonte. Em Prado (2009) foi conduzido um estudo comparando o custo e o *strength* da aplicação dos critérios das técnicas Funcional e Estrutural no paradigma Procedimental e OO.

Apesar das influências causadas pelo paradigma de programação na implementação dos programas Prado não identificou diferenças estatísticas significativas em relação ao custo e a dificuldade de satisfação ao aplicar os critérios da técnica Estrutural nos dois paradigmas.

O critério Análise de Mutantes também pode ser influenciado pelo paradigma de programação durante a atividade de testes. Esta influência é verificada principalmente pelo conjunto de operadores de mutação que é dependente da linguagem de programação. Dada a semelhança de algumas construções sintáticas dessas linguagens com a linguagem C, Vincenzi. (2004) avaliou a aplicabilidade de operadores de mutação desenvolvidos para C no contexto de Java e C++, ou seja, o objetivo foi avaliar quais defeitos modelados pelos operadores de mutação desenvolvidos para C também modelavam defeitos que poderiam ocorrer em programas OO (Vincenzi *et al.*, 2007).

Este trabalho de mestrado tem como objetivo principal avaliar o impacto causado pelo paradigma de programação no Teste de Mutação. Para isso foi conduzido um estudo experimental avaliando o custo e o *strength* da aplicação do critério AM em programas implementados nos paradigmas Procedimental e OO. Na próxima seção são apresentados os conceitos básicos da

Engenharia de Software Experimental que serviram como guia para o planejamento e condução deste trabalho.

2.6 Fundamentos de Engenharia de Software Experimental

Para atender a demanda e as necessidades da indústria e da academia, diferentes métodos, técnicas, processos e ferramentas são propostos regularmente (Basili, 1993). No entanto, muitas vezes não são apresentadas evidências sobre as soluções propostas, ou seja, não há resultados práticos de sua aplicação ou do avanço em relação a tecnologias já estudadas (Basili *et al.*, 1995). A Engenharia de Software Experimental propõe que os estudos tecnológicos da área sejam realizados a partir de experimentos como uma forma de validar e compartilhar eficientemente as soluções elaboradas pelos pesquisadores (Travassos, 2002).

Nesta seção são apresentados os principais conceitos relacionados a Engenharia de Software Experimental. Esse conceitos foram parcialmente extraídos de Travassos (2002); Wohlin *et al.* (2000b).

2.6.1 Experimentação na Engenharia de Software

A experimentação quando aplicada corretamente, pode ajudar a construir uma base de conhecimento confiável reduzindo incertezas sobre teorias propostas. A avaliação experimental das teorias é importante para acelerar a consolidação de idéias promissoras e transferi-las mais rapidamente para a comunidade. Para garantir a transmissão dos conhecimentos adquiridos com o experimento o empacotamento deve ser planejado e executado com seriedade. Somente com uma transmissão eficiente desse conhecimento pesquisadores terão acesso aos estudos e uma melhor orientação a respeito das conclusões obtidas.

Quanto menor o controle sobre um experimento, menos recursos são necessários para sua execução e mais rapidamente o experimento pode ser executado. Por outro lado, uma avaliação experimental, quando executada sem muito rigor, além da difícil replicação, pode colocar em risco a validade das conclusões obtidas. A literatura define três principais tipos de experimentos, diferenciando-os em relação ao controle de execução, o controle de medição, o custo de investigação e a facilidade de repetição. São eles: pesquisa de opinião, estudo de caso e experimento controlado (Travassos, 2002). Para decidir qual o tipo de experimento a ser executado, o pesquisador deve identificar a rigorosidade necessária para o experimento e os recursos disponíveis para sua execução.

A **pesquisa de opinião**, também conhecida como *Survey* é utilizada como uma forma de avaliação de uma técnica proposta, ou seja, é uma investigação executada em retrospectiva. Esse tipo de pesquisa pode ser realizado para a determinação de atributos e características dos itens estudados. A pesquisa de opinião auxilia os pesquisadores a compreender as razões que levaram um grupo de pessoas a utilizar determinada técnica em relação à outra.

A principal forma de coleta de dados para esse tipo de experimento é por meio de questionários ou entrevistas. Por ser um método com baixo custo de aplicação a quantidade de participantes pode ser grande, aumentando assim a validade dos resultados estatísticos. Por outro lado, o *Survey* não oferece nenhum controle sobre a execução ou medição dos dados, por isso é considerado o método menos rigoroso para a realização de experimentos.

O **estudo de caso** é comumente utilizado na monitoração de técnicas e ferramentas. Nessa estratégia as observações são concentradas em um atributo e no seu relacionamento com atributos diferentes (Travassos, 2002). Uma aplicação muito comum para os estudos de caso é na validação de uma nova tecnologia, em que a tecnologia proposta é comparada com a tecnologia atual em relação aos aspectos de interesse e suas vantagens são detalhadas.

Assim como o *survey* o estudo de caso não tem controle sobre a execução do experimento. No entanto, esse método é mais rigoroso do que o a pesquisa de opinião pois a medição das variáveis é controlada. Um ponto importante a ser observado no estudo de caso é a ocorrência de fatores de confusão (*confouding factors*). Esses fatores estão presentes em experimentos em que mais de um fator pode ocasionar um mesmo efeito e devido a falta de controle sobre a execução do experimento não é possível determinar qual a origem do efeito obtido. Estudos de casos devem ser planejados com cuidado a fim de evitar a ocorrência de fatores de confusão pois esses podem prejudicar as conclusões obtidas pelo experimento.

A estratégia mais rigorosa a ser aplicada é o **experimento controlado**. Geralmente esse tipo de experimento é realizado em laboratório, podendo ser *in-vitro*, onde todas as variáveis são controladas, ou *in-vivo*, sob condições naturais. O objetivo do experimento controlado é manipular uma ou algumas variáveis e manter as outras fixas, medindo o efeito do resultado (Travassos, 2002).

Experimentos controlados permitem grande controle sobre a execução medição e investigação da proposta. Para obter tal nível de rigor, é importante que quando possível, todos os fatores para a execução do experimento sejam isolados. O custo para a realização do experimento controlado é maior do que o estudo de caso, porém, devido à maior rigorosidade, ele é mais fácil de ser repetido. A validade do experimento e a confiança agregada é também maior no experimento controlado do que nas outras estratégias.

2.6.2 Terminologia e Conceitos Básicos

A condução de experimentos não é uma tarefa simples, pois é preciso lidar com uma grande quantidade de informações. É importante também que os pesquisadores responsáveis pelo experimento tenham conhecimento sobre o vocabulário utilizado. A seguir são apresentados os principais termos da Engenharia de Software Experimental de acordo com Travassos (2002).

As **variáveis** em um experimento são as informações que são manipuladas pelos pesquisadores e podem ser classificadas em dois grupos dependendo da forma como são obtidas. **Variáveis independentes** são aquelas referentes à entrada do processo de experimentação. Elas correspondem às causas que podem afetar o resultado do experimento. Essas variáveis também são chamadas de **fatores** e seu valor é denominado **tratamento**.

As variáveis que apresentam o efeito causado pelos fatores do experimento são denominadas **variáveis dependentes**, ou seja, são variáveis referentes à saída. O valor das variáveis dependentes é chamado **resultado** e é obtido após a execução do experimento.

O relacionamento entre as variáveis é detalhado na Figura 2.2. Na parte superior as hipóteses são formuladas tentando prever os diversos comportamentos do experimento, construindo assim, um relacionamento causa-efeito. Durante a execução do experimento os valores das variáveis independentes produzem os resultados das variáveis dependentes, resultados esses que devem ser comparados com os efeitos previstos na preparação do experimento.

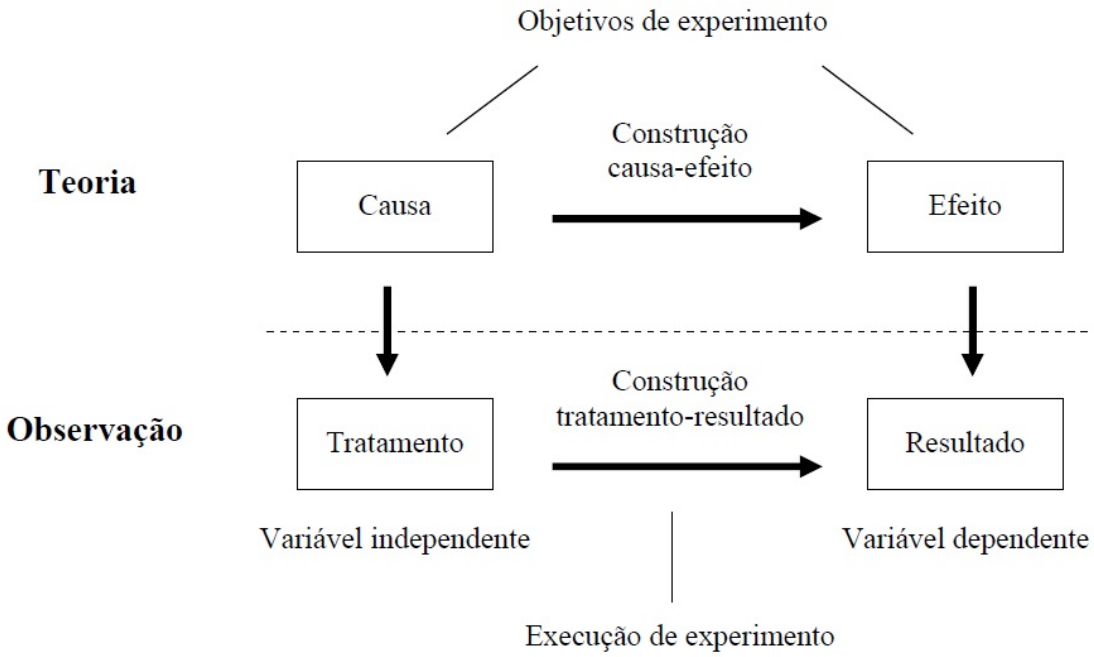

Figura 2.2: Apresentação dos conceitos de um experimento (Wohlin *et al.*, 2000b).

Os **objetos** são todos os artefatos que podem ser manipulados durante o processo de experimentação. O sistema de medição e as diretrizes de execução são responsáveis por definir como os objetos serão manipulados possibilitando a instrumentação do experimento.

Os **participantes** são os indivíduos selecionados dentre a população para conduzir o experimento (Travassos, 2002). A escolha dos participantes deve ser feita de forma cuidadosa para que seja selecionada uma amostra representativa da população a ser estudada. Para atingir essa representatividade alguns aspectos devem ser considerados, como por exemplo, o método de escolha e a quantidade de participantes. Aspectos esses que devem ser sempre proporcionais a variedade da população.

O **contexto do experimento** refere-se às condições de execução do experimento, por exemplo, se o experimento será executado sob condições controladas dentro de um laboratório ou se será executado em condições naturais. No contexto também são definidos quem serão os responsáveis pela condução do experimento, qual a natureza do problema tratado e se o experimento é válido apenas em um contexto particular ou se pode ser generalizado para outros domínios.

O objetivo do experimento é de avaliar a proposta do autor, geralmente expressa como uma relação de causa e efeito. Para tal avaliação o experimento é formulado por meio de hipóteses. A hipótese principal se chama **hipótese nula** e declara que não há nenhum relacionamento estatisticamente significante entre a causa e o efeito (Travassos, 2002). O experimento é realizado com o objetivo de, baseando-se nos resultados obtidos, chegar a conclusão de que a hipótese nula possa ser rejeitada. Para chegar a essa conclusão é importante que alguns princípios sejam seguidos.

2.6.3 Princípios e Validade

Para garantir a confiabilidade e a validade dos resultados do experimento, um conjunto de princípios devem ser considerados. Os princípios definidos na Engenharia de Software Experimental devem ser seguidos durante todo o ciclo de vida do experimento desde o processo de organização, de execução até a fase de apresentação e empacotamento dos resultados. Wohlin *et al.* (2000b) descreve os três princípios básicos da Engenharia de Software Experimental:

Aleatoriedade: todos os métodos estatísticos usados para analisar os dados exigem que a observação seja feita a partir das variáveis independentes e aleatórias (Neto *et al.*, 2001). A aleatoriedade é aplicada na alocação dos objetos, dos participantes e na ordem em que os testes são executados. Ela é utilizada para minimizar alguns efeitos indesejados que podem ocorrer devido à ordenação proposital dos artefatos.

Agrupamento: se existe uma variável independente que tenha um efeito sobre o resultado mas esse efeito não é interessante para os pesquisadores o agrupamento deve ser utilizado. Agrupar variáveis em blocos é uma possível solução para "diluir" efeitos indesejados causados por essas variáveis (Travassos, 2002).

Balanceamento: princípio que procura garantir que todos os tratamentos possuem o mesmo número de participantes com mesmo nível de conhecimento. Essa característica é importante para simplificar a análise estatística e garantir a uniformidade entre os grupos de participantes (Neto *et al.*, 2001).

Além dos princípios básicos apresentados, outros princípios também devem ser considerados, como por exemplo o impacto da investigação nos objetos. Se a investigação de um objeto no experimento pode afetar o seu comportamento, esse impacto deve ser considerado e os valores obtidos devem estar calibrados para contornar o impacto causado.

A garantia de validade do experimento deve ser uma preocupação constante para os organizadores. A validade de um experimento pode ser ameaçada por diversos fatores e são classificadas de acordo com a origem e o impacto causado (Travassos, 2002):

Validade de conclusão: preocupa-se em garantir que o relacionamento entre o tratamento e os resultados do experimento foram construídos de maneira correta. Para tal avaliação é preciso analisar como ocorreu o processo desde a coleta das medidas até as conclusões obtidas. Para a validade de conclusão, conceitos como a escolha do teste estatístico, a escolha do tamanho do conjunto de participantes e a confiabilidade das medidas devem ser consideradas.

Validade interna: verifica se os resultados obtidos são realmente consequência do tratamento que foi aplicado. Essa validação é importante pois os resultados podem ser influenciados por fatores que não foram medidos ou controlados durante o experimento. O foco da verificação da validade interna são nos participantes envolvidos no experimento. Deve ser levado em consideração o aprendizado do participante ao realizar o experimento, o nível de cansaço apresentado no decorrer das atividades, as habilidades individuais e a representatividade em relação à população estudada.

Validade de construção: tem como objetivo garantir que a construção do experimento foi feita de forma correta, ou seja, que tanto o tratamento dado quanto os resultados obtidos refletem bem a causa e o efeito definidos no experimento. Fatores que podem ameaçar a validade de construção podem ser causados tanto pelos participantes quanto pelos pesquisadores.

Os participantes podem ser influenciados pelas hipóteses levantadas no experimento, ou ainda, devido a situação do experimento, eles podem se sentir avaliados e se comportarem de maneira diferente de situações normais (Travassos, 2002). Os pesquisadores podem ser tendenciosos projetando os resultados para aqueles esperados para o experimento.

Validade externa: Define as condições que limitam a generalização do experimento para a prática industrial. Os motivos dessas limitações devem ser explicitados, como por exemplo, a pouca representatividade dos participantes ou condições específicas do experimento que não podem ser encontradas na indústria.

2.6.4 Processo de Experimentação

Em um estudo experimental presume-se a realização de diferentes atividades podendo variar em quantidade e complexidade de acordo com as características do estudo. Na Figura 2.3 são apresentadas as cinco fases gerais que podem estar presente em um processo de experimentação (Wohlin *et al.*, 2000b).

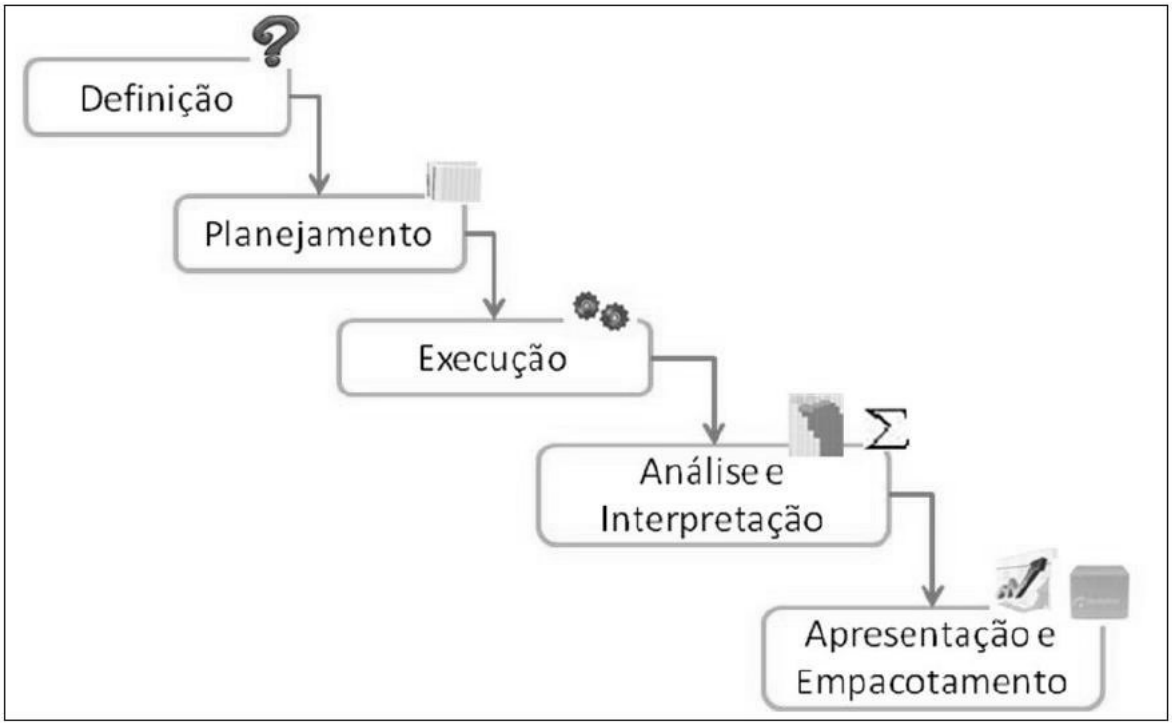

Figura 2.3: Fases gerais de um processo de experimentação (Wohlin *et al.*, 2000b).

O experimento se inicia pela fase de **definição**, nela os pesquisadores descrevem os objetivos, o contexto e o escopo do estudo. Nessa fase são descritas as principais características do experimento fornecendo uma base que posteriormente será utilizada para a formulação das hipóteses.

Na fase de **planejamento** as hipóteses são formuladas de acordo com a relação de causa e efeito descrita pelos pesquisadores. As variáveis que serão analisadas são listadas e as medidas utilizadas para coletá-las são detalhadas. Além da listagem das variáveis os valores que cada

uma delas poderá assumir devem ser apresentados. Durante o planejamento são definidos também o cronograma do experimento e o modo como os recursos serão alocados. É nessa fase também que os participantes são selecionados. Além do cuidado em selecionar uma parcela representativa da população, as características dos participantes podem ser descritas para permitir a identificação e avaliação dos efeitos gerados por cada grupo. Por último os pesquisadores analisam os aspectos de validade e aplicam as ações necessárias para que o experimento não seja comprometido.

Depois de definidos e planejados todos os aspectos do experimento os responsáveis podem avançar para a fase de **execução**. Os fatores mais importantes desta fase são os fatores humanos que podem prejudicar o experimento. Para tanto, os participantes devem ser informados sobre a metodologia utilizada e as atividades que cada um deve realizar. Além disso, deve haver um esforço para tranquilizá-los de modo que suas reações sejam as mais próximas de uma situação normal. Com o consentimento dos participantes o experimento é iniciado e os resultados podem ser coletados. A coleta dos dados deve ser realizada de maneira que não cause efeito significativo ao processo sendo estudado (Travassos, 2002).

Os dados obtidos na fase de execução geralmente precisam de um tratamento para que possam ser analisados experimentalmente. Para tanto, na fase de **análise e interpretação**, métodos estatísticos são aplicados com o objetivo de eliminar informações fora da distribuição normal e agrupar os resultados de forma lógica. Com base nas informações obtidas os resultados são analisados e as conclusões descritas. Com o conjunto de dados coletados o pesquisador pode avaliar as hipóteses levantadas na fase de planejamento e se os resultados indicarem, a hipótese nula pode ser rejeitada.

É importante que na fase de análise e interpretação estejam documentados os passos que levaram os pesquisadores as conclusões obtidas. Devem ser documentadas a descrição dos métodos estatísticos utilizados bem como os princípios seguidos para garantir a validade do experimento .

A última fase do processo de experimentação é a de **apresentação e empacotamento**. Nessa fase todas as informações relevantes a respeito da execução do experimento são documentadas. Também são descritas as hipóteses definidas pelos pesquisadores, as atividades executadas, as dificuldades encontradas, os resultados e experiências realizados, as características dos participantes, o ambiente em que o experimento foi aplicado entre outros. Todos esses artefatos devem ser documentados e se possível disponibilizados para a comunidade permitindo assim a verificação e replicação do experimento.

Com a replicação do experimento os pesquisadores adquirem o conhecimento adicional a respeito dos conceitos estudados e podem avaliar os novos resultados comparando-os com

resultados do experimento original. De qualquer maneira, o aumento das repetições possibilita o aumento do aprendizado dos conceitos investigados e também a calibração das características do experimento (Travassos, 2002).

O estado de arte atual a respeito de empacotamento de experimentos indica a ausência de normas internacionais aprovadas. Essa falta de padronização permite que cada pesquisador defina quais artefatos descrever e muitas vezes omite informações importantes durante o empacotamento. Amaral e Travassos (2003) definem quatro aspectos fundamentais que devem ser tratados durante a apresentação e empacotamento de um experimento:

- A comunidade do processo do experimento: devem ser descritas todas as pessoas que colaboraram de alguma forma para a execução do experimento, sejam os pesquisadores responsáveis pelo experimento, os indivíduos que participaram da fase de execução e outros usuários que participaram ou utilizaram os resultados do experimento para estudos ou fins industriais.

- A organização do experimento: descreve todas as informações de como o experimento foi planejado, como os participantes foram preparados, como ocorreu a instrumentação e quais as diretrizes para a execução do experimento.

- Os artefatos do experimento: todos os artefatos utilizados para a coleta dos resultados devem ser documentados e se possível disponibilizados para a comunidade. Os artefatos dependem do tipo do experimento executado, mas podem ser desde algoritmos, código-fonte ou módulos executáveis e até itens que ajudaram na coleta da informação.

- Os resultados do experimento: a apresentação dos resultados deve conter além dos dados que permitiram a conclusão do experimento, os dados originais obtidos a partir da execução, ou seja, os dados puros, sem a aplicação dos métodos estatísticos. Também devem ser descritas as informações sobre o processo de experimentação e os problemas encontrados pelos pesquisadores durante o experimento.

2.7 Estudos Teóricos e Experimentais de Critérios de e Técnicas de Teste

Como visto na Seção 2.3, diversas técnicas e critérios foram propostas para guiar a atividade de teste. O testador deve definir qual critério ou qual combinação utilizar de acordo com a natureza do sistema, o risco de falhas aceito pelo cliente e o tempo disponível para testes. Para auxiliar

essa decisão estudos teóricos e experimentais são realizados procurando comparar os critérios existentes, fornecendo assim, informações sobre o custo, dificuldade de satisfação e eficácia em revelar erros.

Estudos **teóricos** procuram analisar as características dos critérios com o objetivo de estabelecer uma hierarquia entre eles. Segundo Maldonado *et al.* (2007), a comparação entre os critérios, em geral, é feita por meio da relação de inclusão. Um critério de teste C_1 inclui um critério C_2 se, para qualquer programa P e qualquer conjunto de casos de teste T_1 C_1-adequado, T_1 for também C_2-adequado, e para algum conjunto de casos de teste T_2 C_2-adequado, T_2 não for C_1-adequado.

Com base nessa relação Rapps e Weyuker (1985) analisaram a hierarquia dos critérios de fluxo de dados. A relação obtida é ilustrada na Figura 2.4. Ao analisá-la, pode-se identificar que o critério Todas-Arestas inclui o critério Todos-Nós, ou seja qualquer conjunto de casos de teste que satisfaça o critério Todas-Arestas também satisfaz o critério Todos-Nós. Quando não é possível estabelecer essa ordem de inclusão para dois critérios, diz-se que esses critérios são incomparáveis, como é o caso dos critérios Todas-Defs e Todas-Arestas (Maldonado *et al.*, 2007).

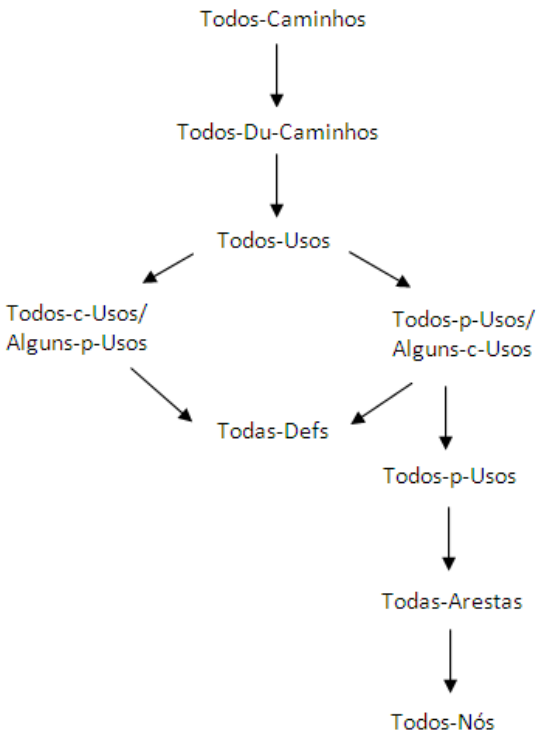

Figura 2.4: Relação de Inclusão dos critérios de fluxo de dados (Rapps e Weyuker, 1985).

Ao definir a família de critérios Potenciais-Usos, Maldonado (1991) verificou a relação de inclusão entre estes e os critérios existentes para a técnica Estrutural. Foi constatado que o critério Todos-Potenciais-Usos inclue o critério Todos-Usos e mesmo na presença de caminhos não executáveis é possível estabelecer uma hierarquia entre os critério Todas-Arestas e Todos-Caminhos.

Apesar dos estudos teóricos fornecerem informações valiosas em relação à hierarquia de alguns critérios e também sobre custos de aplicação, as vezes é impossível estabelecer a relação de inclusão entre eles como é o caso dos critérios Todas-Definições e Todas-Arestas.

Segundo Weyuker *et al.* (1991), o uso de estudos teóricos para escolha de técnicas e critérios de teste podem apresentar informações de valor limitado em algumas situações práticas, e que o uso de informações probabilísticas obtidas a partir de experimentos poderiam contornar algumas dessas limitações. Bertolino (2004) defende a complementaridade entre esses estudos, para ele, os estudos teóricos serviriam para levantar algumas evidências sobre em quais condições uma técnica de teste seria melhor do que outra e os resultados observados a partir de estudos experimentais serviriam para mostrar quando (e em qual contexto) tais condições são satisfeitas.

Com o objetivo de avaliar experimentalmente medidas de custo e satisfação dos critérios de testes, estudos **experimentais** são aplicados comparando diferentes técnicas e critérios. Assim, critérios que não podem ser comparados de forma teórica podem ter seu custo e eficácia avaliados experimentalmente.

Segundo Maldonado *et al.* (1998), três aspectos costumam ser avaliados do ponto de vista de estudos experimentais: custo, eficácia em revelar defeitos e dificuldade de satisfação (*strength*).

- **custo:** métrica que avalia o esforço necessário para a geração e execução dos casos de teste para cobrir determinado critério. O custo pode ser calculado levando em consideração diversos fatores como o número de casos de teste construídos, o número de elementos requeridos, o número de elementos requeridos não executáveis, o tempo gasto para execução dos testes, entre outros.

- **eficácia:** medida que revela a capacidade de um critério em revelar erros. Uma forma comum de medir a eficácia de um conjunto de casos de teste é inserir erros no programa e ao aplicar o critério de teste analisar a quantidade de erros identificada pelo critério. Avaliar a eficácia de um critério é uma atividade bastante difícil pois a inserção de erros propositalmente nos programas pode não representar erros comuns cometido por programadores, por outro lado, analisar a quantidade de erros encontrados sem conhecer a quantidade total de erros no programa pode prejudicar a avaliação estatística dos resultados.

- **dificuldade de satisfação (*strength*):** medida que estabelece a probabilidade de um conjunto de caso de teste T_1 adequado para o critério C_1 ser adequado também para o critério C_2, onde $C_1 \neq C_2$, ou seja, verificar o quanto se consegue satisfazer um critério C_2 tendo satisfeito o critério C_1 (Wong, 1993).

Como pode ser observado, a maioria das métricas utilizadas depende de alguma forma dos casos de teste construídos para satisfazer o critério. É importante ressaltar que ao adotar esta abordagem alguns cuidados devem ser tomados em relação à complexidade dos casos de teste construídos. Enquanto um testador pode implementar casos de teste simples que cubram apenas um elemento requerido por caso de teste, outro testador pode desenvolver casos de teste mais complexos que cubram vários elementos de uma só vez. Para que o número de casos de teste seja uma medida válida para a avaliação da complexidade de critérios a implementação dos casos de teste deve seguir as mesmas diretrizes.

Feitas essas observações, são apresentados alguns estudos experimentais que têm como objetivo avaliar diferentes técnicas e critérios de testes em relação às métricas apresentadas:

Com o objetivo de comprovar experimentalmente a hierarquia de critérios definidas para a técnica Estrutural, Weyuker (1990) conduziu um estudo experimental avaliando o custo dos critérios Todos-Usos, P-Usos, C-Usos e DU-Caminhos. Neste estudo foi avaliado o custo de aplicação dos critérios da técnica Estrutural, baseando-se na cardinalidade do conjunto de casos de teste. Os testes foram executados na ferramenta ASSET (Frankl e Weyuker, 1985) sobre 29 programas implementados em Pascal. Os resultados mostram que o conjunto de casos de teste adequados ao critério Todos-P-Usos, em geral, também satisfazem o critério Todos-C-Usos. Foi constatado também que o conjunto de casos de teste adequado ao critério Todos-Usos frequentemente foram adequados também ao critério Todos-DU-Caminhos. Além disso foi proposta uma maneira de prever a quantidade de casos de teste necessária para a satisfação dos critérios a partir das características do programa a ser testado.

Devido ao reconhecido custo da aplicação do critério AM, Mathur e Wong (1993) avaliaram experimentalmente duas técnicas para a seleção de um subconjunto de operadores de mutação: a mutação aleatória (selecionando 10% de cada operador de mutação) e a mutação restrita (selecionando um subcojunto de operadores de mutação). O objetivo do estudo era avaliar o custo e o benefício das duas estratégias. Os resultados obtidos mostraram que ambos os critérios mostraram-se igualmente eficazes e com um custo baixo (sem perdas de eficácia consideráveis) em relação ao critério AM considerando todos os operadores.

Wong *et al.* (1995) também compararam experimentalmente os critérios Análise de Mutantes e Todos-Usos em relação ao custo e *strength*. Os resultados obtidos revelaram que o *strength* do critério Análise de Mutantes se mostrou maior do que do critério Todos-Usos, pois

o conjunto de casos de teste adequado ao critério Análise de Mutantes foi também adequado ao critério Todos-Usos. Porém o conjunto de casos de teste adequado ao critério Todos-Usos não se mostrou adequado para o Teste de Mutação para todos os programas avaliados.

Frankl *et al.* (1996) também compararam o teste de mutação com o critério Todos-Usos. Foi usado um conjunto inicial de nove programas implementados em Pascal contendo erros naturais. Como a ferramenta de teste escolhida para auxiliar o critério AM foi desenvolvida para programas implementados em Fortran, cada programa foi traduzido de Pascal para Fortran com o cuidado de que essa tradução causasse o menor impacto possível. Os resultados mostraram que o teste de mutação foi mais efetivo em cinco programas, o critério Todos-Usos em dois e nos outros dois não foram observadas diferenças.

Em Souza *et al.* (1997) foi realizado um estudo experimental comparando o critério Análise de Mutantes com a família de critérios Potenciais-Usos. Foi avaliado o custo e o *strength* dos critério em programas implementados em C. Os resultados mostram que o custo do teste de mutação foi maior do que dos critérios da família Potenciais-Usos. Também foi identificado que mesmo do ponto de vista experimental o critério Análise de Mutantes e Todos-Potenciais-Usos são incomparáveis.

Em Barbosa *et al.* (1998) foi proposto um procedimento para a determinar um conjunto Essencial de operadores de mutação para a linguagem C. Com o procedimento proposto é possível, por meio de uma maneira sistemática e criteriosa selecionar os operadores de mutação mais eficientes e dessa forma reduzir os custos da aplicação do critério Análise de Mutantes. É importante ressaltar também que devido à flexibilidade do critério e do procedimento proposto, o procedimento Essencial (responsável por identificar os operadores essenciais) pode ser aplicado ao critério AM em outras linguagens de programação.

Um estudo experimental comparando teste aleatório, Funcional e Análise de Mutantes, foi realizado em Linkman *et al.* (2003). O experimento contou com seis participantes, os quais já possuíam conhecimento sobre os critérios e o programa utilizado. O objetivo do experimento era avaliar a adequação do teste aleatório e Funcional em relação ao critério Análise de Mutantes. Para o teste aleatório, foram selecionados os sete conjuntos de teste gerados em Wong (1993). Para o teste Funcional Sistemático foram gerados 76 casos de teste. O critério Análise de Mutantes foi usado para avaliar a eficácia dos conjuntos de teste gerados. Com base em todos os operadores de mutação para programas em C, foram gerados 4.624 mutantes, dos quais 335 foram identificados como equivalentes. Os resultados obtidos no estudo revelam em geral, os escores de mutação atingidos pelos conjuntos gerados para a técnica Funcional, mostraram-se maiores do que os gerados para o teste aleatório. Os resultados permitiram concluir que, ainda que sejam necessários estudos de casos para garantir validade estatística, os dados obtidos for-

necem indícios de que os testes Funcionais podem atingir um alto grau de cobertura, quando aplicados corretamente.

Em Smith e Williams (2007) foi conduzido um estudo experimental com o objetivo de avaliar e categorizar os operadores da ferramenta MuJava (Offutt *et al.*, 2006) utilizando a MuClipse (Smith e Williams, 2009). Foram utilizadas duas versões de um conjunto de três classes de uma aplicação Web denominada iTrust implementada em Java. Os resultados obtidos mostram que os operadores condicionais COR, COI e COD geraram uma quantidade significante de mutantes "uteis", ou seja, mutantes que exigiram a construção de casos de teste específicos para diferenciá-los do programa original. Também foi identificado que os operadores AOIS, JSI, JID e PCD produziram a maior quantidade de mutantes equivalentes, ou mutantes que eram mortos com muita facilidade. Além disso diversos operadores não foram gerados pelas características do código-fonte (AODU,AODS,SOR, LOR, LOD e ASRS no nível de método e AMC, IHI, IHD, IOD, IOP, IOR, ISI, ISD, IPC, PNC, PMD, PPD, PCC, OMR, OMD, OAN, JTI, JTD, EOA e EOC no nível de classe). O teste de mutação foi executado juntamente com uma ferramenta chamada dJUnit que indica quais linhas do código-fonte foram executadas. Com isso os pesquisadores puderam observar quais características do código-fonte não são cobertas pelos operadores de mutação utilizados. Foi identificado que a maioria dos blocos que tratavam exceções não foram executados, o que indica a necessidade de operadores para este conjunto de instruções. Também foram encontrados alguns trechos em que algumas expressões com o comando *if* não foram executadas, devido a erros existentes no programa original que não permitiam alcançar tais instruções.

Também comparando o Teste de Mutação com a técnica Estrutural, em Li *et al.* (2009) foi conduzido um estudo experimental comparando o custo e a eficácia entre os critérios AM e os critérios estruturais de Fluxo de Dados e Fluxo de Controle. O estudo envolveu um conjunto de 29 classes implementadas em Java. Os conjuntos de CT adequados a cada critério foram construídos manualmente e verificados com o auxílio da ferramenta MuJava (Offutt *et al.*, 2006) para o teste de mutação. Para a técnica Estrutural, foi utilizada uma ferramenta web de análise de cobertura em grafos (Ammann *et al.*, 2008). Erros foram inseridos manualmente nos programas e cada critério teve sua eficácia avaliada pela quantidade de erros revelados pelos conjuntos de CT adequados. O custo de cada critério foi medido por meio da quantidade de casos de teste necessários para a satisfação do critério. Os resultados indicaram que o critério com mais elementos requeridos foi o AM. No entanto ele foi o que necessitou a menor quantidade de casos de teste, uma vez que grande parte dos mutantes gerados foram mortos facilmente. Foi constatado também que a eficácia do teste de mutação foi a maior entre os critérios estudados.

Devido a falta de estudos comparando critérios de teste em diferentes paradigmas, em seu trabalho de mestrado Prado (2009) avaliou o custo e *strength* da técnica Estrutural nos paradig-

mas Procedimental e OO utilizando o conjunto de 32 programas implementados em C e em Java extraídos dos livros de Algoritmos e Estruturas de Dados (Ziviani, 2005a,b). Foram avaliados os critérios baseados em Fluxo de Dados e Fluxo de Controle e os resultados obtidos fornecem indícios de que não existe diferença significativa entre o custo e o *strength* da técnica Estrutural entre os dois paradigmas.

2.8 Considerações Finais

Neste capítulo foram apresentados os fundamentos do teste de software, abordando as técnicas e critérios mais utilizados. Foram apresentadas também algumas ferramentas desenvolvidas para apoiar a execução dos testes em diversos critérios e linguagens de programação. As principais diferenças em relação ao paradigma Procedimental e OO foram enumeradas e os impactos causados por essas diferenças foram detalhados. Também foram apresentados os fundamentos e conceitos básicos da área de Engenharia de Software Experimental e alguns estudos teóricos e experimentais avaliando diferentes critérios de testes em diferentes paradigmas de programação.

Com todos os conceitos e fundamentos apresentados é possível compreender o contexto do experimento e todos os passos necessários para sua execução. A partir das informações apresentadas neste capítulo o experimento pode ser planejado e definido.

CAPÍTULO

3

Definição e Planejamento do Estudo

3.1 Considerações Iniciais

De acordo com Travassos (2002), a definição e o planejamento de um experimento representam as primeiras etapas do estudo e são nessas fases em que são definidas as metas e os objetivos. Também são detalhadas cada uma das tarefas a serem executadas, os artefatos envolvidos e os subprodutos a serem gerados. A definição e o planejamento apresentados neste Capítulo foram construídos com base na estrutura de processos definida por Wohlin *et al.* (2000a).

Na Seção 3.2 são apresentados os objetivos e as metas do experimento. Na Seção 3.3 são descritos o contexto, as hipóteses, os participantes e as variáveis do experimento. Também são apresentadas as atividades propostas para a execução e avaliação do estudo. Por último, são descritos os princípios de validade e como eles foram aplicados no estudo apresentado.

3.2 Definição

Na fase de definição, o experimento é expresso em termos dos problemas e objetivos. Nesta fase, são descritas as principais questões que deram origem ao experimento. A partir dessas questões, são formuladas as hipóteses, o escopo é definido e o experimento é planejado (Travassos, 2002).

3.2.1 Objetivos

As questões que motivaram este experimento foram:

- O custo da aplicação do critério AM varia de acordo com o paradigma no qual o programa é implementado?
- Quando um conjunto de caso de teste (CT) é adequado a um critério em um paradigma, ele é também adequado ao mesmo critério em um programa com as mesmas funcionalidades implementado em outro paradigma?
- O quanto um conjunto de CT adequado aos critérios da técnica Funcional são adequados ao critério AM em programas no paradigma Procedimental e OO?
- O quanto um conjunto de CT adequado aos critérios da técnica Estrutural são adequados ao critério AM em programas no paradigma Procedimental e OO?
- Qual o custo da execução do Teste de Mutação aplicando uma estratégia incremental considerando o paradigma Procedimental e OO?
- Qual o custo e qual e *strength* da aplicação do critério AM considerando o conjunto com todos os operadores da Proteum em relação ao conjunto de operadores Essenciais (Barbosa, 1998; Barbosa *et al.*, 2001) em programas do domínio de estrutura de dados?

A partir da enumeração das questões, os objetivos podem ser definidos. Geralmente o experimento é composto de um objetivo principal, também denominado objetivo global, e a partir deste são definidos os objetivos secundários:

Objetivo global: Este experimento tem como objetivo principal avaliar o impacto causado pelo paradigma de programação no custo e no *strength* do critério AM em programas Procedimentais e OO considerando o domínio de estruturas de dados.

Objetivos secundários: Avaliar o custo e o *strength* do critério AM em relação aos critérios das técnicas Funcionais e Estruturais nos paradigmas Procedimental e OO e Avaliar o custo e o *strength* do critério AM considerando o conjunto de todos os operadores da Proteum em relação ao conjunto de operadores Essenciais (Barbosa, 1998; Barbosa *et al.*, 2001), considerando o domínio de estruturas de dados.

3.2.2 Definição das Metas

As metas e objetivos foram definidos seguindo a estrutura apresentada em (Travassos, 2002). Para tanto é necessário identificar os objetos de estudo, o objetivo, o foco da qualidade, a perspectiva e o contexto do experimento.

Ao definir os **objetos do estudo** são especificadas as entidades que serão estudadas ao longo do experimento:

- Critério de teste AM no paradigma Procedimental;
- Critério de teste AM no paradigma OO;
- Critérios de teste da técnica Estrutural no paradigma Procedimental;
- Critérios de teste da técnica Estrutural no paradigma OO;
- Critérios de teste da técnica Funcional no paradigma Procedimental;
- Critérios de teste da técnica Funcional no paradigma OO.

O **propósito** define a intenção da realização do experimento. Este experimento tem o propósito de avaliar o impacto causado pelo paradigma de programação na aplicação dos critérios de teste.

O **foco de qualidade** indica os principais aspectos da qualidade que estão sendo estudados: Custo e Dificuldade de Satisfação (*strength*).

A **perspectiva** especifica o ponto de vista que os resultados do experimento serão interpretados. A perspectiva do experimento é do ponto de vista de pesquisadores.

O **contexto** especifica o ambiente onde o experimento está sendo executado: o experimento foi executado por um pesquisador, sobre um conjunto de programas do domínio de estrutura de dados utilizando ferramentas para auxiliar a aplicação do critério AM nos paradigmas Procedimental e OO.

3.3 Planejamento

De acordo com Travassos (2002) o planejamento do experimento serve como roteiro para a condução, execução e análise do assunto investigado. Nesta fase, acontecem a formulação das hipóteses, a definição das variáveis e a seleção dos participantes. O contexto do experimento é descrito com mais detalhes e o escopo é restringido. Também são descritos o projeto do experimento e as considerações relacionadas à validade do estudo. O resultado desta fase apresenta o experimento totalmente elaborado e pronto para execução.

3.3.1 Seleção do contexto

O estudo foi realizado pelo próprio pesquisador que o definiu. Foi utilizado um conjunto de programas de estruturas de dados, implementados em C (representando o paradigma Procedimental) e Java (representando o paradigma OO). Esse programas foram extraídos dos livros "Projeto de Algoritmos com Implementações em Pascal e C"e "Projeto de Algoritmos com Implementações em Java e C++"de Ziviani (Ziviani, 2005a,b). Para a execução dos testes de mutação foram utilizadas as ferramentas Proteum (Delamaro, 1993) e MuClipse (Smith e Williams, 2009) para programas em C e Java, respectivamente. Essas ferramentas foram selecionadas devido à estabilidade e maturidade para a aplicação do critério AM nos dois paradigmas. Como o conjunto de operadores de mutação implementados na Proteum é maior do que o da MuClipse, foi considerado também o conjunto de operadores Essenciais (Barbosa, 1998; Barbosa *et al.*, 2001) definido com o objetivo de reduzir os custos do critério AM sem prejudicar a eficácia. Esse conjunto seleciona apenas 11 dos 75 operadores implementados na Proteum.

A avaliação do *strength* do critério AM em relação aos critérios pertencentes às técnicas Funcionais e Estruturais foi executada a partir de um conjunto de CT construído com Prado (2009). O conjunto de casos de teste adequado à técnica Funcional foi construído de forma incremental. Primeiramente foram construídos os conjuntos de casos de teste adequados ao critério Particionamento em Classes de Equivalência e a partir dele, foi construído o conjunto de CT adequado ao critério Análise de Valor Limite. Para a construção do conjunto de casos de teste adequados à técnica Estrutural, foi utilizado inicialmente o conjunto de casos de teste adequados à técnica Funcional. A partir desse conjunto, foram construídos novos CT para se adequarem aos critérios Todos-Nós, Todas-Arestas, Todos-Usos e Todos-Potenciais-Usos também de maneira incremental. Para este estudo foram considerados o conjunto de CT adequados ao critério Análise de Valor Limite, representando a técnica Funcional, e o critério Todos-Potenciais-Usos representando a técnica Estrutural.

3.3.2 Hipóteses

O estudo executado busca identificar características pouco exploradas no domínio de teste de software. Desse modo as hipóteses propostas não pretendem ser definitivas, mas sim fornecer evidências sobre as questões estudadas e que posteriormente possam ser utilizadas como referência para novos estudos na área.

A primeira métrica utilizada para avaliar o impacto do paradigma nos critérios de teste é o custo. Admitindo-se que o custo da aplicação do critério AM sobre um conjunto de programas seja medido em termos da quantidade de CT necessária para a satisfação do critério, quantidade

de mutantes gerados e quantidade de mutantes equivalentes identificados, acredita-se que neste estudo possam ser evidenciadas as diferenças de custo relacionadas ao paradigma procedimental e OO. Para avaliar essas variáveis foram definidas as seguintes hipóteses:

Hipótese 1

> **Hipótese Nula:** Não existe diferença de custo do critério AM entre o paradigma A (Procedimental) e B (OO).
>
> H_0: Custo(A) = Custo(B)
>
> **Hipótese Alternativa:** Existe diferença de custo do critério AM entre o paradigma A (Procedimental) e B (OO).
>
> H_1: Custo(A) $\neq$ Custo(B)

Do mesmo modo, admitindo-se que o *strength* do critério AM no paradigma A em relação ao paradigma B possa ser medido por meio do escore de mutação obtido pelo conjunto de CT adequado a B, quando aplicado contra os mutantes gerados no paradigma A, acredita-se que o *strength* do critério AM no paradigma Procedimental em relação ao paradigma OO possa ser avaliado comparando o escore de mutação obtido pelo conjunto de CT adequados ao paradigma OO, quando executados contra os mutantes gerados no paradigma Procedimental e vice-versa. Com isso, a seguinte hipótese pode ser definida:

Hipótese 2

> **Hipótese Nula:** Não existe diferença de *strength* entre o critério AM no paradigma A (Procedimental) e B (OO).
>
> H_0: Strength(A:B) = Strength(B:A)
>
> **Hipótese Alternativa:** Existe diferença de *strength* entre o critério AM no paradigma A (Procedimental) e B (OO).
>
> H_1: Strength(A:B) $\neq$ Strength(B:A)

Aproveitando o conjunto de programas estudados no experimento, também foi avaliado o custo e o *strength* do conjunto com todos os operadores da Proteum em relação ao conjunto de operadores Essenciais. Para avaliar esses valores foi construída a seguinte hipótese:

Hipótese 3

Hipótese Nula: Não existe diferença de custo entre o critério de teste AM com o conjunto de operadores Essencial (A) e o conjunto com todos os operadores (B) no paradigma Procedimental.

H_0: Custo(A) = Custo(B)

Hipótese Alternativa: Existe diferença no custo do critério de teste AM, entre o conjunto de operadores Essencial (A) e o conjunto com todos os operadores (B) no paradigma Procedimental.

H_1: Custo(A) $\neq$ Custo(B)

Além disso pretende-se avaliar o escore de mutação obtido pelos conjuntos de casos de teste adequados ao critério AM considerando o conjunto de operadores Essencial em relação ao conjunto com todos os operadores da Proteum. Pretende-se também avaliar o *strength* do critério AM em relação aos critérios das técnicas Funcional e Estrutural nos dois paradigmas.

3.3.3 Seleção dos Indivíduos

Como o estudo em questão exige o conhecimento das diferentes técnicas de teste e bastante tempo de dedicação, o experimento foi realizado pelo próprio pesquisador que o definiu. O estudo anterior, em que foram preparados o ambiente de teste, a descrição e especificação dos programas e a construção dos casos de teste adequados à técnica Funcional foi executado em conjunto com outro aluno de mestrado no grupo (Prado, 2009). Prado também construiu e executou os testes Estruturais para o mesmo conjunto de programas e comparou o custo e o *strength* da aplicação dos critérios baseados em Fluxo de Controle e em Fluxo de Dados nos dois paradigmas.

Seria conveniente utilizar no estudo, um número maior de indivíduos para que possíveis diferenças relacionadas à construção de casos de teste pudessem ser identificadas. Espera-se que a documentação gerada neste estudo possa servir como base para que o experimento possa ser replicado com um número maior de participantes.

3.3.4 Variáveis

Variáveis Independentes: Travassos (2002) define variáveis independentes como aquelas que podem ser manipuladas ou controladas no processo de experimentação. Foram identificadas neste experimento as variáveis:

1. Paradigma em que os programas foram implementados;
2. Ferramentas de teste utilizadas;
3. Conjuntos de operadores de mutação utilizado;
4. Tamanho e complexidade dos programas;
5. Habilidade do testador em aplicar corretamente os critérios.

O foco do estudo é na variável "Paradigma em que os programas foram implementados", entretanto, como já explicado anteriormente, a ferramenta de teste e os operadores de mutação nela definidos podem influenciar significativamente nos resultados obtidos. Como cada ferramenta trabalha com um conjunto distinto de operadores de mutação, o estudo não pode avaliar o impacto do paradigma de implementação sem considerar a ferramenta de teste utilizada.

Também foi analisada a variável "Conjunto de operadores de mutação utilizados" para o paradigma Procedimental. Essa variável foi analisada para avaliar o custo e o *strength* do conjunto com todos os operadores da Proteum em relação ao conjunto dos operadores Essenciais (Barbosa, 1998; Barbosa *et al.*, 2001).

A variável "Tamanho e complexidade dos programas" não foi considerada, pois para a comparação das métricas foram avaliados programas semelhantes nos dois paradigmas, as possíveis diferenças de complexidade que possam existir entre as versões do programa nos dois paradigmas são decorrentes da maneira de abstrair a solução, influenciada principalmente pelo paradigma de implementação.

As diferenças relacionadas à habilidade do testador não foram consideradas, pois o mesmo testador foi responsável por aplicar os testes nos dois paradigmas. Além disso, a ordem em que os testes foram executados foi selecionada de forma aleatória com o objetivo de diminuir a influência humana no experimento.

Variáveis Dependentes: são aquelas nas quais são observados os resultados da manipulação das variáveis independentes, ou seja, as variáveis que expressam o efeito na relação causa-efeito definida no experimento (Travassos, 2002). As variáveis dependentes identificadas neste estudo considerando o conjunto de operadores das ferramentas Proteum e MuClispe nos dois paradigmas foram:

1. Quantidade de mutantes gerados para o critério AM no paradigma Procedimental (operadores Essenciais e todos operadores) e OO (QMUT).
2. Quantidade de mutantes equivalentes identificados para o critério AM no paradigma Procedimental (operadores Essenciais e todos operadores) e OO (QEQUIV).

3. Quantidade de casos de teste necessários para a satisfação do critério AM no paradigma Procedimental (operadores Essenciais e todos operadores) e OO a partir do conjunto de CT adequados à técnica Funcional (QCTF).

4. Quantidade de casos de teste necessários para a satisfação do critério AM no paradigma Procedimental (operadores Essenciais e todos operadores) e OO a partir do conjunto de CT adequados às técnicas Funcional e Estrutural (QCTE).

5. Escore de mutação obtido pelos conjuntos de CT Funcional-Adequados no critério AM no paradigma Procedimental (operadores Essenciais e todos operadores) e OO (EFEP, EFTP, EFOO).

6. Escore de mutação obtido pelos conjuntos de CT Funcional+Estrutural-Adequados no critério AM no paradigma Procedimental (operadores Essenciais e todos operadores) e OO (EEEP, EETP, EEOO).

7. Escore de mutação obtido pelos conjuntos de CT adequados ao critério AM no paradigma OO em relação ao critério AM no paradigma Procedimental (operadores Essenciais e todos operadores) (EOOEP, EOOTP).

8. Escore de mutação obtido pelos conjuntos de CT adequados ao critério AM no paradigma Procedimental (operadores Essenciais e todos operadores) em relação ao critério AM no paradigma OO (EEPOO, ETPOO).

9. Escore de mutação obtido pelos conjuntos de CT adequados ao critério AM considerando os operadores Essenciais em relação ao conjunto com todos os operadores da Proteum (EEPTP).

3.3.5 Descrição da instrumentação do experimento

Conforme apresentado anteriormente o objetivo principal do experimento é avaliar o impacto causado pelo paradigma de programação no teste de mutação. Desse modo para cada um dos 32 programas avaliados deve ser construído um conjunto de CT adequado ao critério AM a partir do conjunto de CT adequados à técnica Funcional e outro a partir do conjunto de CT adequados à técnica Estrutural. A aplicação do critério AM nos programas implementados em C deve seguir duas perspectivas: uma considerando apenas o conjunto Essencial de operadores de mutação (Barbosa, 1998; Barbosa *et al.*, 2001) e outra considerando todos os operadores definidos na Proteum (Delamaro e Maldonado, 2001).

Durante a atividade de construção do conjunto de CT adequados ao critério AM, cada programa foi avaliado para funcionar corretamente nas ferramentas utilizadas e em seguida os mutantes gerados. Todos os mutantes equivalentes identificados devem ser descritos para posterior análise. Após a geração dos mutantes e identificação daqueles equivalentes, os conjuntos de casos de teste adequados à técnica Funcional e Estrutural devem ser executados nas ferramentas de apoio ao critério AM para a avaliação do *strength* . Para avaliar o *strength* entre o critério AM nos dois paradigmas os conjuntos de CT adequados critério AM no paradigma Procedimental devem ser executados contra os mutantes gerados no paradigma OO e o escore de mutação analisado. O mesmo processo deve ser repetido para avaliar o *strength* do critério AM no paradigma Procedimental em relação ao critério AM no OO considerando os dois conjuntos de operadores.

Por último, para a avaliação do *strength* do conjunto com todos os operadores da Proteum em relação ao conjunto de operadores Essencial no paradigma Procedimental, o conjunto de CT adequados ao critério AM considerando os operadores Essenciais devem ser executados contra os mutantes gerados por todos os operadores da Proteum. A partir do escore de mutação obtido o *strength* deve ser analisado.

3.3.6 Validade

Durante a avaliação de validade do experimento são considerados os riscos que podem comprometer os resultados e conclusões obtidas no experimento. As questões de validade de um experimento podem ser classificadas em quatro tipos:

A validade de conclusão corresponde à habilidade de chegar a uma conclusão correta sobre os relacionamentos entre o tratamento e os resultados obtidos no experimento. Uma das ameaças a validade de conclusão é a confiabilidade no processo de identificação das métricas. Neste estudo, as métricas avaliadas foram extraídas automaticamente pelas ferramentas de testes e pela aplicação sistemática do critério AM, com isso acredita-se que a repetição do experimento por diferentes pesquisadores produzirá resultados semelhantes aos obtidos neste estudo. Além disso, a avaliação estatística foi feita por métodos adequados, verificando sempre a normalidade do conjunto de dados. Foi aplicado o teste t de Student(Hollander e Wolfe, 1999) para conjuntos com distribuição de dados normal e o teste não-paramétrico de Wilcoxon (*Signed-Rank*)(Hollander e Wolfe, 1999) para as distribuições não-normais, ambos com nível de significância $(\alpha) = 0.05$. A execução do experimento por apenas um pesquisador oferece um risco a validade de conclusão, pois este pode influenciar nas métricas que dependem do fator humano, como por exemplo a quantidade de casos de testes necessárias para a satisfação

do critério e o *strength* obtido por esses casos de testes. A replicação do experimento com uma quantidade maior de participantes minimizaria os riscos relacionados a validade de conclusão.

A validade interna depende da existência de uma relação causal entre tratamento e resultado. Logo, para garantir que este princípio não seja violado, os responsáveis pelo experimento devem garantir que não há interferência de algum fator que não tenha sido medido ou controlado, na relação causa-efeito. Um fator que poderia comprometer a validade interna do experimento é o aprendizado do testador na construção dos casos de teste. Para minimizar os efeitos deste fator, os programas foram selecionados de forma aleatória, além disso os testes foram construídos seguindo sistematicamente o critério AM, ou seja, cada caso de teste foi construído com o objetivo de matar um mutante específico relativo a uma determinada implementação e o escore de mutação atualizado em seguida.

Para garantir a validade de construção o pesquisador deve assegurar que o tratamento reflete a construção da causa, e o resultado a construção do efeito. Para isso foram selecionadas ferramentas de teste de mutação conhecidas na comunidade. Além disso a aplicação sistemática do critério AM tende a diminuir a influência causada por fatores humanos. A repetição do experimento com um número maior de participantes é interessante para avaliar as possíveis diferenças nos resultados, a partir de casos de teste construídos por diferentes pessoas.

A validade externa do experimento refere-se à capacidade de generalização dos resultados obtidos. Como o escopo do estudo se restringiu a programas do domínio de estrutura de dados e o conjunto de programas dentro desse domínio é bastante abrangente, o conjunto de programas não ameaça a validade do estudo para esse domínio. Entretanto, como as duas ferramentas avaliadas utilizam um conjunto distinto de operadores de mutação, os resultados de custo e dificuldade de satisfação obtidos neste estudo não podem ser generalizados para o paradigma Procedimental e OO sem que seja levada em consideração o conjunto de operadores implementados por cada uma das ferramentas utilizadas.

3.4 Considerações Finais

Neste capítulo foram apresentadas a definição e o planejamento do estudo seguindo os conceitos de Engenharia de Software Experimental apresentados no Capítulo 2. A definição e planejamento do experimento são importantes para detalhar como o experimento foi concebido e quais as atividades planejadas para a avaliação das hipóteses definidas. No Capítulo 4 é detalhado o processo de preparação e execução do experimento.

CAPÍTULO

4

Preparação do Ambiente e Execução do Experimento

4.1 Considerações Iniciais

Após a definição e o planejamento do estudo a próxima etapa no processo de experimentação é a execução. Esta é uma etapa muito delicada no experimento pois possui uma maior influência do fator humano. A coleta dos dados deve ser realizada de maneira que não cause efeito significativo no processo sendo estudado. Nesta etapa também é realizada a validação preliminar dos dados experimentais.

Neste capítulo são apresentados os passos realizados para a execução do experimento. São detalhadas desde a preparação das variáveis até a avaliação dos resultados. É mostrado como o planejamento do experimento foi executado, as dificuldades encontradas e as decisões tomadas para dar continuidade ao experimento. Um programa do conjunto estudado foi selecionado para exemplificar as atividades realizadas, ilustrando todos os passos do experimento, de modo a facilitar sua replicação futura.

Na Seção 4.2 é detalhado como foi feita a configuração do ambiente no qual o experimento foi executado. Também é descrita a estrutura e organização da documentação gerada. A Seção 4.3 exemplifica cada passo realizado no experimento, as dificuldades encontradas e as soluções aplicadas.

4.2 Preparação

Para que o processo de execução do experimento pudesse ser realizado, foi necessária a preparação do ambiente de testes. Como este trabalho é complementar ao trabalho de Prado (2009), optou-se por dar continuidade ao padrão já adotado.

4.2.1 O Ambiente de Testes

O ambiente construído em conjunto com Prado para a execução do experimento avaliando os critérios das técnicas Funcional e Estrutural constitui-se de uma máquina virtual com todos os artefatos necessários para a replicação do experimento. Além da configuração do Sistema Operacional para a execução das ferramentas de teste, a máquina virtual contém:

- As ferramentas de testes utilizadas no estudo devidamente instaladas e configuradas;
- A documento de especificação de cada programa analisado;
- A documentação da execução de cada critério de teste executado para cada programa; e
- Conjunto de artefatos gerados por cada sessão de teste executada em cada critério para cada programa.

Dentre os benefícios encontrados com a adoção da máquina virtual podem-se citar: a portabilidade, não só das ferramentas e do sistema operacional, mas de toda a configuração de sistema, a possibilidade de replicação e geração de *backup* de forma prática e relativamente rápida, a viabilidade de evolução de configurações de ferramentas sem o comprometimento de uma configuração já estável, a possibilidade de interrupção e continuação de uma tarefa em datas diferentes mantendo-se o mesmo estado do sistema operacional e dos programas utilizados e a persistência direta dos resultados gerados internamente pela cópia em uso (Prado, 2009).

O ambiente construído para a avaliação dos critérios de testes nos paradigmas Procedimental e OO possui as seguintes características:

- Sistema operacional Linux, distribuição Ubuntu - versão 7.04 (Feisty Fawn), com a versão 4.1.2 do compilador GCC instalada e configurada;
- IDE Eclipse para desenvolvedores C/C++ - versão 3.4.0 (Ganymede), o qual já possui o plug-in da ferramenta JUnit versão 3.8.2 instalado, para aplicação dos testes funcionais em Java;

- Plug-in da ferramenta "C/C++ Unit Testing Easier"(Cute) versão 1.6.1 instalado sobre a plataforma Eclipse, para aplicação dos testes funcionais em C;
- Ferramenta JaBUTi versão 1.0, para aplicação dos testes estruturais em Java;
- Ferramenta Poke-Tool, com variáveis de ambiente carregadas durante a inicialização do Sistema Operacional, para aplicação dos testes estruturais em C; e
- Ferramenta CodeAnalyserPro versão 1.1, para coleta das medidas de implementação dos programas nas duas linguagens(LOC e Complexidade Ciclomática).

A esse ambiente foram adicionadas e configuradas as ferramentas Proteum (Delamaro, 1993) e MuClipse (Smith e Williams, 2009) para apoiar o teste de mutação nos dois paradigmas.

4.2.2 Estrutura de Diretórios

Também foi aproveitada a estrutura de diretórios e arquivos definidas em conjunto com Prado para disponibilização/armazenamento dos artefatos de cada programa utilizado no experimento. A configuração original disponibiliza na raiz do diretório de cada programa os seguintes documentos e formulários de teste:

- Documento de especificação do programa: nesse documento é descrito como deve ser o funcionamento do algoritmo apresentado. Os termos utilizados para exemplificar a execução do algoritmo são definidos, assim como os requisitos que devem ser cumpridos na sua implementação. Por último são descritas todas as operações que o programa deve ser capaz de realizar. Esse documento é independente da implementação do programa, servindo como um guia para a construção do mesmo em diferentes linguagens de programação;
- Documento de implementação do programa em C: documento que descreve como a especificação foi convertida para um programa em C. Nele são detalhadas a estrutura do código do programa e a interface das operações implementadas. São descritas também algumas métricas relacionadas a implementação, como a quantidade de Linhas de Código, Complexidade de McCabe (McCabe e H., 1996), quantidade de funções/procedimentos, etc.;
- Documento de implementação do programa em Java: documento que descreve como a especificação foi convertida para um programa em Java seguindo o mesmo modelo adotado para a documentação de implementação do programa em C ;

- Documento de Classe de Equivalência: documento que descreve as classes de equivalência construídas a partir da especificação dos programas. Esse documento também é independente da implementação e foi construído a partir da especificação do programa para auxiliar a construção dos conjuntos de casos de teste adequados aos critérios da técnica Funcional nos dois paradigmas;

- Documento de teste do programa em C: documento que descreve a atividade de teste do programa, cobrindo o teste Funcional e Estrutural. Neste documento são descritos a quantidade de elementos requeridos, a quantidade de elementos requeridos não-executáveis e a quantidade de CT necessários para a adequação à cada um dos critérios estudados;

- Documento de teste do programa em Java: documento que descreve a atividade de teste do programa implementado em Java, seguindo o mesmo modelo do documento de testes do programa em C;

- Diretório dos programas fontes: diretório contendo os programas originais divididos em dois sub-diretórios: C e Java;

- Diretório dos testes Funcionais: diretório contendo as sessões de testes dos critérios da técnica Funcional divididos em dois sub-diretórios de acordo com a ferramenta utilizada: "Cute" para o teste funcional em C e "Junit" para o teste em Java; e

- Diretório dos testes Estruturais: diretório contendo as sessões de testes dos critérios da técnica Estrutural divididos em dois sub-diretórios de acordo com a ferramenta utilizada: "Poke - Tool" para o teste em C e "JaBUTi" para o teste em Java.

Os documentos de teste de cada linguagem foram atualizados com informações referentes ao teste de mutação: quantidade de mutantes gerados, a quantidade e descrição dos mutantes equivalentes identificados e a quantidade de casos de teste adicionais.

Além dos dados referentes à execução do teste de mutação, foram adicionados aos documentos informações sobre a cobertura obtida pelos conjuntos de casos de teste adequados às técnicas Funcional e Estrutural.

Também foi adicionado um diretório contendo as sessões de testes relacionadas aos testes do critério AM divididas em três sub-diretórios de acordo com a ferramenta e o conjunto de operadores utilizados: "Essenciais-Proteum" e "Todos-Proteum" para o teste de mutação em C e "MuClipse" para o teste em Java.

Para análise das hipóteses do experimento foi criado um diretório específico na raiz do diretório de testes dividido em cinco sub-diretórios, cada um avaliando uma característica em particular.

- OperadoresEssenciaisCtestesJava: Conjunto de sessões de teste de cada programa, onde é avaliado o escore obtido pelo conjunto de CT adequados ao critério AM em Java em relação aos mutantes gerados pelo conjunto dos operadores Essenciais da ferramenta Proteum;

- OperadoresTodosCtestesJava: Conjunto de sessões de teste de cada programa, onde é avaliado o escore obtido pelo conjunto de CT adequados ao critério AM em Java em relação aos mutantes gerados pelo conjunto com todos os operadores da ferramenta Proteum;

- OperadoresJavatestesEssenciaisC: Conjunto de sessões de teste de cada programa, onde é avaliado o escore obtido pelo conjunto de CT adequados ao critério AM Procedimental considerando os operadores Essenciais em relação ao critério AM no paradigma OO;

- OperadoresJavatestesTodosC: Conjunto de sessões de teste de cada programa, onde é avaliado o escore obtido pelo conjunto de CT adequados ao critério AM Procedimental considerando todos os operadores em relação ao critério AM no paradigma OO; e

- EscoreEssenciais: Conjunto de sessões de teste de cada programa, onde é avaliado o escore obtido pelo conjunto de CT adequados ao critério AM procedimental considerando os operadores Essenciais em relação aos mutantes gerados pelo conjunto com todos os operadores da ferramenta Proteum.

4.2.3 Descrição dos Programas

Como já citado anteriormente o estudo utiliza um conjunto de especificações extraídas de (Ziviani, 2005a,b). Este conjunto possui 32 especificações de programas do domínio de estruturas de dados. Para cada especificação foi testada a versão implementada em C (considerando os dois conjuntos de operadores) e em Java. Para a auxiliar a compreensão desta atividade, os programas foram identificados seguindo a Tabela 4.1. Na tabela são apresentadas também duas métricas relacionadas à complexidade dos programas testados: quantidade de linhas de código (LOC) e complexidade ciclomática (CC) (McCabe e H., 1996) para cada programa implementado em C e em Java respectivamente.

Pode-se observar que a complexidade ciclomática é bastante semelhante nas versões em C e em Java dos programas. Já a quantidade de linhas de código, aparentemente, mostrou-se um pouco maior nos programas implementados no paradigma procedimental. Também pode ser observado que a complexidade dos programas avaliados não é uniforme. Enquanto alguns programas realizam apenas uma operação simples, como por exemplo identificar o maior valor em um conjunto, outros programas implementam uma estrutura de dados completa contendo

Tabela 4.1: Descrição dos programas usados no estudo

Id	Descrição	C		Java	
		LOC	**CC.**	**LOC**	**CC.**
1	Obtém o valor máximo de um conjunto.	12	3	8	3
2	Obtêm os valores máximo e mínimo pelo método 1.	14	4	13	4
3	Obtêm os valores máximo e mínimo pelo método 2.	14	4	13	4
4	Obtêm os valores máximo e mínimo pelo método 3.	27	9	25	9
5	Ordenação de vetor de inteiros.	17	4	14	4
6	Calcula a sequência de Fibonacci (recursivo).	6	2	7	2
7	Calcula a sequência de Fibonacci (iterativo).	9	2	11	2
8	Obtêm o valor máximo e mínimo de forma recursiva.	20	5	20	5
9	Ordenação pelo método Mergesort.	23	6	22	8
10	Custo mínimo para multiplicação de "n"matrizes.	25	7	31	7
11	Estrutura "lista"por meio de arranjos.	44	10	28	13
12	Estrutura "lista"por auto-referência.	42	5	19	5
13	Estrutura "pilha"por meio de arranjos.	36	7	23	7
14	Estrutura "pilha"por auto-referência.	49	6	32	6
15	Estrutura "fila"por meio de arranjos.	39	8	29	8
16	Estrutura "fila"por auto-referência.	50	7	38	7
17	Ordenação por seleção, inserção, shellsort, quicksort e heapsort.	107	31	86	32
18	Fila de prioridades por meio de arranjos.	76	17	54	19
19	Ordenação Parcial.	117	35	88	35
20	Pesquisa sequencial e binária em vetor de inteiros.	52	11	30	11
21	Arvore binária.	94	25	74	29
22	Hashing usando lista encadeada.	118	28	64	19
23	Hashing usando Endereçamento Aberto.	98	28	72	29
24	Grafo usando matriz de adjacência.	116	28	74	27
25	Grafo usando lista de adjacência com auto-referência.	76	8	73	24
26	Grafo usando lista de adjacência com arranjos.	123	23	82	27
27	Busca em profundidade em um Grafo.	76	8	36	9
28	Busca em largura em um Grafo.	199	23	51	15
29	Obtêm os componentes fortemente conectados de um grafo.	130	13	51	14
30	Algoritmo de Prim para descoberta da árvore geradora mínima.	199	23	51	27
31	Casamento exato para busca em cadeias de caracteres.	68	24	47	25
32	Casamento aproximado para busca em cadeias de caracteres.	38	7	26	7

diversas operações como é o caso da implementação da árvore binária. Como a especificação é a mesma nos dois paradigmas a diferença entre a complexidade dos programas não foi considerada no experimento.

4.3 Execução

Para exemplificar a etapa de execução do experimento é utilizado o Programa 21 que implementa uma Árvore Binária. A execução do experimento foi dividida em diversas fases que serão apresentadas a seguir.

4.3.1 Preparação

A primeira atividade realizada foi a verificação da adequação de todos os programas nas ferramentas e as correções de eventuais problemas de compatibilidade com as mesmas.

A MuClipse não exigiu alteração nos programas testados, pois ela utiliza a JUnit como ferramenta de apoio a construção e execução dos casos de teste. A ferramenta Proteum exige algumas características no código-fonte que não estavam presentes em todos os programas. Esses programas foram adaptados para contemplar as exigências da ferramenta, preocupando-se sempre em minimizar o impacto causado pelas alterações. As principais alterações realizadas nos programas implementados em C para funcionarem corretamente na Proteum foram:

- Exclusão de comentários de linha: Quando o programa continha algum comentário do tipo "//" a ferramenta não conseguia instrumentar o programa corretamente, para isso, todos os comentários desse tipo foram excluídos.

- Alterações no retorno de alguns métodos: A ferramenta utiliza os resultados gerados na saída padrão para distinguir os mutantes gerados do programa original e assim determinar quais estão vivos e quais estão mortos. Alguns métodos não retornavam os valores computados dessa forma. Esses foram alterados para exibirem seus resultados na saída padrão.

Como a construção de casos de teste não são padronizadas em C do mesmo modo que em Java, algumas alterações foram necessárias para que os casos de teste pudessem ser executados na Proteum. Seguindo o padrão utilizado pela CUTE, foi construída uma função ASSERT que verifica uma expressão e, caso a mesma apresente um valor verdadeiro, imprime "OK" na saída padrão e, caso contrário, imprime "ERRO". Desse modo, a ferramenta pode diferenciar um mutante do programa original por meio da saída gerada pela função de verificação ASSERT.

Outra dificuldade encontrada ao utilizar a Proteum, foi em relação a interface para a inserção dos casos de teste. A ferramenta espera que o programa a ser testado receba os parâmetros dos CT por linha de comando, o que não acontecia nas implementações estudadas. Como os CT construídos eram significativamente complexos, não foi possível utilizar esta interface diretamente. Para solucionar este problema os CT foram construídos de forma semelhante aos padrão adotado pela JUnit ou seja, em forma de uma função que declara as variáveis necessárias e invoca a função a ser testada. Para que o testador pudesse ter um controle sobre a execução dos CT, foi construído no arquivo de testes a função "Main" que recebe como parâmetro o argumento passado por linha de comando ao programa, e de acordo com o valor passado, executa o CT selecionado. Assim para testar o programa da Árvore Binária foi criado o arquivo ConjuntoTeste.c ilustrado no Programa 4.1.

Programa 4.1: Exemplo de implementação dos Casos de Testes para a linguagem C

```
1 #include "arvorebinaria.c"
2 void ASSERT(int exp) {
3   if (exp)
4      printf("OK\n");
5   else
6     printf("ERRO\n");
7 }
8
9 void test_1_1() {
10         ...
11 }
12 .
13 .
14 .
15 int main(int argc,char** argv )
16 {
17   int teste;
18   teste = atoi(argv[1]);
19   printf("\n%d: ",teste);
20
21   if (teste==1)
22    test_1_1();
23   else if (teste==2)
24    test_2_1();
25   .
26         .
27   .
28   else if (teste==14)
29    test_4_3_S();
30
31 }
```

Para a instanciação de uma nova sessão de teste o código do programa a ser testado foi sempre o ConjuntoTeste.c que é responsável por incluir o programa a ser testado e executar os CT nele implementados. A Proteum permite selecionar quais funções devem ser testadas, assim somente as funções pertencentes ao código testado foram selecionados para a geração dos mutantes e teste. Uma das desvantagens da utilização desta estratégia é que toda vez que o critério exigir a construção de mais casos de teste o programa deve ser novamente instrumentado, uma vez que a implementação do programa está vinculada à implementação dos casos de teste.

Para automatizar esta tarefa foi criado um *script* responsável por instrumentar o programa, gerar os mutantes de acordo com os operadores selecionados, marcar os mutantes já identificados como equivalentes e executar os CT construídos contra os mutantes restantes. Com isso a estratégia de inserir os CT no programa testado pode ser adotada sem perda de eficiência.

4.3.2 Aplicação do critério Análise de Mutantes

Após a adequação de todos os programas nas ferramentas dos dois paradigmas, iniciou-se a aplicação do critério Análise de Mutantes. Para reduzir o impacto causado pelo aprendizado por parte do testador, a ordem em que os programas foram testados e a ordem do conjunto de operadores utilizados foi determinada de forma aleatória. Para a construção dos casos de teste foi utilizada uma estratégia incremental, tomando como base o conjunto de CT adequados à técnica Funcional e Estrutural. A ordem do conjunto inicial também foi selecionada de forma aleatória para cada programa.

Para o programa árvore binária a seguinte ordem foi sorteada:

Conjunto de Operadores de Mutação:

1. Operadores Essenciais C.
2. Todos Operadores Java;
3. Todos Operadores C;

Conjunto CT inicial:

1. Conjunto de CT adequado à técnica Funcional e Estrutural; e
2. Conjunto de CT adequado à técnica Funcional.

Pelo sorteio, a primeira sessão de testes a ser avaliada para o programa Árvore Binária foi a que considera os operadores Essenciais (Barbosa, 1998; Barbosa *et al.*, 2001) do paradigma Procedimental tomando como conjunto de casos de teste inicial o conjunto adequado aos critérios da técnica Funcional e Estrutural.

Para este conjunto de operadores foram gerados 196 mutantes, dos quais 49 permaneceram vivos após a execução do conjunto de CT inicial. Para exemplificar esta atividade o Programa 4.2 detalha a função "Retira" extraída do programa Árvore Binária implementada em C. Esta função é responsável pela eliminação de um registro na Árvore Binária ou, caso o registro não exista, uma mensagem de erro é retornada. A função recebe como parâmetro os dados do registro a ser retirado e um apontador para a raiz da árvore.

Programa 4.2: Implementação da função Retira do programa "Árvore Binária" em C.

```
1 void Retira(Registro x, Apontador *p)
2 { Apontador Aux;
3   if (*p == ((void *)0))
4   { printf("Erro : Registro nao esta na arvore\n");
```

```
5       return;
6     }
7     if (x.Chave < (*p)->Reg.Chave)
8     { Retira(x, &(*p)->Esq); return; }
9     if (x.Chave > (*p)->Reg.Chave)
10    { Retira(x, &(*p)->Dir); return; }
11    if ((*p)->Dir == ((void *)0))
12    { Aux = *p;
13      *p = (*p)->Esq;
14      free(Aux);
15      return;
16    }
17    if ((*p)->Esq != ((void *)0))
18    { Antecessor(*p, &(*p)->Esq);
19      return;
20    }
21    Aux = *p;
22    *p = (*p)->Dir;
23    free(Aux);
24  }
```

Dos 49 mutantes sobreviventes ao conjunto inicial de casos de teste, 15 deles foram identificados como equivalentes. Como exemplo é apresentado o mutante 142 ilustrado no Programa 4.3 que apresenta uma alteração na função Retira na linha 11. Este mutante não pode ser distinguido do programa original pelo fato da variável $(*p)- > Dir$ só assumir valores maiores ou iguais a zero, já que se trata de um apontador para uma posição na árvore. Desse modo a condição $if((((*p)- > Dir) <= ((void(*))0)))$ só será satisfeita quando $(*p)- > Dir$ for igual a zero, que é a mesma condição do programa original.

Além da identificação de cada mutante, foi anotado também a função na qual ocorreu a mutação: "Retira", a linha: 11 e o operador responsável pela geração do mutante: ORRN. Além de documentar a atividade de testes, esses dados fornecem informações importantes que posteriormente podem ser utilizadas em uma análise mais detalhada em relação aos operadores de cada paradigma como por exemplo a análise da porcentagem de mutantes equivalentes gerados por cada operador.

Programa 4.3: Exemplo de mutante equivalente em C.

```
1  void Retira(Registro x, Apontador *p)
2  { Apontador Aux;
3    if (*p == ((void *)0))
4    { printf("Erro : Registro nao esta na arvore\n");
5      return;
6    }
7    if (x.Chave < (*p)->Reg.Chave)
8    { Retira(x, &(*p)->Esq); return; }
9    if (x.Chave > (*p)->Reg.Chave)
10   { Retira(x, &(*p)->Dir); return; }
11   if ( (((* p)->Dir) <= ((void (*)) 0) ) )
```

```
12   { Aux = *p;
13     *p = (*p)->Esq;
14     free(Aux);
15     return;
16   }
17   if ((*p)->Esq != ((void *)0))
18   { Antecessor(*p, &(*p)->Esq);
19     return;
20   }
21   Aux = *p;
22   *p = (*p)->Dir;
23   free(Aux);
24 }
```

O mutante 189 ilustrado no Programa 4.4 é um exemplo de mutante sobrevivente ao conjunto de casos de teste adequados às técnicas Funcional e Estrutural que exigiu a construção de novos casos de teste para que pudesse ser distinguido do programa original. Este mutante aplica a função SUCC sobre a variável $x.Chave$ na linha 7, essa função considera que naquele ponto é avaliado o sucessor do valor apresentado, ou seja $x.Chave + 1$.

Para evidenciar a diferença entre o mutante e o programa original foi necessário construir um caso de teste em que a condição $if((SUCC((x.Chave)) < (((*p) - > Reg).Chave)))$ não fosse satisfeita no programa mutante mas que fosse satisfeita no programa original. Para isso foi criado um caso de teste que retirasse o elemento 31 da árvore binária construída a partir da seguinte sequência de inserção: 32,31. Com este CT ao avaliar a linha 7 na primeira iteração do algoritmo recursivo, a chave buscada (SUCC(31)) não será menor que a chave da raiz da árvore (32). Como a condição da linha 9, que verifica se a chave buscada (31) é maior do que a chave da raiz (32), também não será satisfeita, o algoritmo assume que a chave a ser retirada é a apontada por p, de valor 32 e na sequência da função retira esse item da árvore binária.

Programa 4.4: Exemplo de mutante não equivalente em C.

```
1  void Retira(Registro x, Apontador *p)
2  { Apontador Aux;
3    if (*p == ((void *)0))
4    { printf("Erro : Registro nao esta na arvore\n");
5      return;
6    }
7    if (SUCC(x.Chave) < (*p)->Reg.Chave)
8    { Retira(x, &(*p)->Esq); return; }
9    if (x.Chave > (*p)->Reg.Chave)
10   { Retira(x, &(*p)->Dir); return; }
11   if ( (((* p)->Dir) == ((void (*)) 0) ) )
12   { Aux = *p;
13     *p = (*p)->Esq;
14     free(Aux);
15     return;
16   }
17   if ((*p)->Esq != ((void *)0))
```

```
18    { Antecessor(*p, &(*p)->Esq);
19      return;
20    }
21    Aux = *p;
22    *p = (*p)->Dir;
23    free(Aux);
24  }
```

Após a execução do teste de mutação, foram identificados 15 mutantes equivalentes. Com esse valor foi possível calcular o quanto o conjunto de CT adequados às técnicas Funcional e Estrutural satisfazem o critério AM considerando este conjunto de operadores. Este cálculo foi feito por meio do escore de mutação obtido que foi de 0,812.

O custo da aplicação do critério AM tomando como base o conjunto de CT adequado à técnica Funcional e Estrutural também foi avaliado. Na Tabela 4.2 esses valores podem ser conferidos. Na primeira coluna é apresentado o identificador do programa, seguido da quantidade de mutantes gerados e da quantidade de mutantes equivalentes identificados. A quantidade de CT iniciais é apresentada na quarta coluna, seguido da quantidade de mutantes vivos após a execução desses CT e do escore de mutação obtido. Por último são mostrados a quantidade de CT adicionais, o total de CT construídos e quantos desses foram efetivos para o teste de mutação. É importante comparar a quantidade de CT adicionais com a quantidade de CT efetivos, uma vez que nem todos os conjuntos de CT construídos para os critérios das técnicas Funcional e Estrutural foram significativos para critério AM.

Tabela 4.2: Execução do critério AM a partir do conjunto de CT adequados às técnicas Funcional e Estrutural considerando os operadores Essenciais da Proteum.

Prog	Mut	Equiv.	CT ini.	M. Vivos	Escore	CT adic.	Total CT	CT Efetivos
21	196	15	15	34	0,812	10	25	19

O mesmo procedimento foi executado para avaliar o custo da execução do critério AM a partir do conjunto de CT adequado à técnica Funcional. Como foi avaliado o mesmo programa com o mesmo conjunto de operadores a geração dos mutantes e a determinação dos mutantes equivalentes foi a mesma da etapa anterior. Sendo assim para obter o escore de mutação bastou aplicar o novo conjunto de casos de teste em uma nova sessão de testes. Para o conjunto adequado à técnica Funcional sobraram ainda 59 mutantes (contra os 49 do conjunto adequado à técnica Estrutural), para atingir 100% do escore foram necessários contruir mais 11 casos de teste. Na tabela 4.3 são apresentados os resultados da aplicação do teste de mutação a partir do conjunto de casos de teste adequados à técnica Funcional.

Tabela 4.3: Execução do critério AM a partir da de CT adequados à técnica Funcional considerando os operadores Essenciais da Proteum.

Prog	Mut	Equiv.	CT ini.	M. Vivos	Escore	CT adic.	Total CT	CT Efetivos
21	196	15	14	44	0,757	11	25	19

Seguindo a sequência definida no sorteio a próxima atividade foi testar o programa no paradigma OO ilustrado no Programa 4.5. Como dito anteriormente, não foi necessária nenhuma alteração no programa original. Os mutantes foram gerados pela ferramenta MuClipse utilizando todos os 43 operadores de mutação e em seguida os mutantes restantes analisados.

Programa 4.5: Árvore Binária implementada em Java.

```
1  package cap5;
2  import cap4.Item; // @{\it vide Programa~\ref{prog:interfaceitem}}@
3  public class ArvoreBinaria {
4    private static class No {
5      Item reg;
6      No esq, dir;
7    }
8    private No raiz;
9    private void central (No p) {
10     if (p != null) {
11       central (p.esq);
12       System.out.println (p.reg.toString());
13       central (p.dir);
14     }
15   }
16   private Item pesquisa (Item reg, No p) {//
17         ...
18   }
19   private No insere (Item reg, No p) {//
20         ...
21   }
22   private No antecessor (No q, No r) {
23         ...
24   }
25   private No retira (Item reg, No p) {//
26                 ...
27   }
28   public ArvoreBinaria () {
29      this.raiz = null;
30   }
31   public Item pesquisa (Item reg) {
32      return this.pesquisa (reg, this.raiz);
33   }
34   public void insere (Item reg) {
35        this.raiz = this.insere (reg, this.raiz);
36   }
```

```
37   public void retira (Item reg) {
38         this.raiz = this.retira (reg, this.raiz);
39   }
40 }
```

A MuClipse não possui a funcionalidade de marcar um mutante como equivalente, desse modo eles tiveram de ser anotados manualmente e descontados para o cálculo do escore de mutação. O programa Árvore Binária gerou apenas 12 mutantes sendo que 9 deles foram mortos pelo conjunto de CT adequados à técnica Funcional e Estrutural. Dos 3 mutantes restantes 1 deles foi identificado como equivalente. A mutação identificada consiste na exclusão do construtor ArvoreBinaria() na linha 28 que é responsável por inicializar a variável raiz com o valor *default null* e sua exclusão não altera a execução do algoritmo.

Outro mutante gerado adiciona o modificador *static* antes da variável raiz na linha 8. Para distinguir este mutante do programa original foi necessária a construção de um caso de teste que instanciasse dois objetos da classe ArvoreBinaria e preenchesse esses dois objetos com valores distintos. Ao imprimir os valores de cada classe o programa original imprime corretamente enquanto que o mutante imprime duas vezes o último valor preenchido. Este caso de teste matou também o outro mutante restante, totalizando apenas 1 CT adicional para que o critério fosse satisfeito. Sendo assim, os resultados após a aplicação do critério AM a partir do conjunto de CT adequados à técnica Funcional e Estrutural é apresentada na Tabela 4.4.

Tabela 4.4: Execução do critério AM a partir do conjunto de casos de teste adequados à técnica Funcional e Estrutural no paradigma OO.

Prog	Mut	Equiv.	CT ini.	M. Vivos	Escore	CT adic.	Total CT	CT Efetivos
21	12	1	15	2	0,818	1	16	6

Como pode ser observado ao comparar as tabelas de custo nos dois paradigmas a ferramenta em Java gerou um número de mutantes bastante inferior em relação à ferramenta em C, o que pode ser um indício da necessidade de uma avaliação mais detalhada sobre os operadores implementados para este paradigma.

A mesma atividade foi realizada tomando como conjunto de CT inicial o conjunto adequado à técnica Funcional e os resultados são apresentados na Tabela 4.5.

Tabela 4.5: Execução do critério AM a partir do conjunto de CT adequados à técnica Funcional no paradigma OO.

Prog	Mut	Equiv.	CT ini.	M. Vivos	Escore	CT adic.	Total CT	CT Efetivos
21	12	1	14	2	0,818	1	15	6

Por último, foi executado o teste de mutação considerando todos os operadores da Proteum. Como este conjunto engloba o conjunto dos operadores Essenciais, alguns mutantes equivalentes já haviam sido identificados. No entanto, não foi possível identificar uma relação direta entre os mutantes equivalentes dos dois conjunto de operadores. A identificação dessa relação não foi possível devido ao fato da ferramenta utilizar como identificador a ordem em que os mutantes foram gerados. Assim, o mutante 142, já descrito no Programa 4.3, é identificado como 321 quando considerado o conjunto com todos os operadores da Proteum.

Para o programa Árvore Binária considerando todos os operadores da Proteum, foram gerados 440 mutantes sendo que 30 deles foram identificados como equivalentes. Dos 410 mutantes não equivalentes, o conjunto de CT adequados à técnica Funcional e Estrutural matou 372 deles, obtendo um escore mutação de 0,907. Para atingir o escore de 1,00 foram necessários mais 10 casos de teste. Os resultados desta sessão de teste são resumidos na Tabela 4.6.

Tabela 4.6: Execução do critério AM a partir do conjunto de CT adequado à técnica Funcional e Estrutural considerando todos os operadores da Proteum.

Prog	Mut	Equiv.	CT ini.	M. Vivos	Escore	CT adic.	Total CT	CT Efetivos
21	440	30	15	38	0,907	10	25	19

O mesmo processo foi realizado a partir do conjunto de CT adequados à técnica Funcional e os resultados são apresentados na Tabela 4.7.

Tabela 4.7: Execução do critério AM a partir dos conjunto de CT adequado à técnica Funcional considerando todos os operadores da Proteum.

Prog	Mut	Equiv.	CT ini.	M. Vivos	Escore	CT adic.	Total CT	CT Efetivos
21	440	30	14	55	0,866	11	25	19

Ao final desta atividade tem-se como resultado o custo da aplicação do critério AM nos dois paradigmas considerando três conjuntos de operadores diferentes. Pode-se avaliar também a diferença de custo entre os conjuntos de CT inicial (Funcional e Estrutural). Na próxima sessão é detalhado como foi feita a avaliação do *strength* entre os paradigmas.

4.3.3 Análise do *strength*

A avaliação do *strength* foi feita considerando os três conjuntos de operadores. Na sessão anterior era importante que o testador tivesse um conhecimento detalhado sobre a codificação do programa para identificar as diferenças entre o mutante e o programa original e construir casos de teste que evidenciassem essas diferenças. Por isso, a ordem de execução foi realizada horizontalmente, ou seja, cada programa era testado sobre todas as perspectivas (paradigma de programação, conjunto de operadores e conjunto de casos de teste inicial) para que então o testador seguisse para outro programa. Na avaliação do *strength* este conhecimento específico de cada programa não é exigido, sendo assim esta avaliação foi realizada verticalmente, ou seja, cada perspectiva foi analisada por completo (para todos os programas) para que então fosse avaliada uma outra perspectiva.

Primeiramente foi avaliado o *strength* do critério AM considerando o conjunto com todos os operadores em relação ao conjunto dos operadores Essenciais de C. Para isso, foi criado o diretório "EscoreEssenciais". A atividade de geração dos mutantes do programa foi a mesma do teste de mutação considerando todos os operadores. Foi aproveitado portanto o *script* responsável por gerar os mutantes e marcar os mutantes identificados como equivalentes. Para essa sessão foram considerados apenas o conjunto com os CT efetivos.

Assim, para o programa Árvore Binária o conjunto contendo os 19 casos de teste efetivos considerando os operadores Essenciais foram executados contra os 440 mutantes gerados considerando todos os operadores da Proteum. Este conjunto obteve o escore 1,00, ou seja, todos os mutantes não equivalentes foram mortos.

A próxima etapa do experimento foi verificar o quanto o conjunto de casos de teste adequado ao critério AM no paradigma OO satisfaz o critério AM no paradigma Procedimental considerando todos os operadores de mutação. Para isso o conjunto de CT efetivos adequado ao critério AM no paradigma OO foi executado contra os mutantes gerados na Proteum. Para esta verificação também foi criado um diretório específico, denominado "TodosCTestesJava".

Como o programa a ser testado e os casos de teste estavam em diferentes linguagens de programação houve a necessidade de converter os casos de teste de Java para a linguagem C. Esta conversão foi feita cuidadosamente para reduzir o impacto causado pela troca de linguagem e paradigma.

Para o programa Árvore Binária foram aplicados contra os 440 mutantes gerados no paradigma Procedimental o conjunto com os 6 casos de teste efetivos para o mesmo critério no paradigma OO, esse conjunto de casos de teste obteve um escore de mutação de 0,834 pois descontando os 30 mutantes equivalentes sobraram ainda 58 mutantes vivos.

A mesma atividade foi realizada considerando apenas os operadores Essenciais do paradigma Procedimental. Esse conjunto com 6 CT quando aplicados contra os 196 mutantes gerados no paradigma Procedimental considerando apenas os operadores Essenciais obteve um escore de 0,740, sobrando 47 mutantes vivos. Pode-se observar que o escore de mutação obtido pelo conjunto de casos de teste adequados ao critério AM em Java foi menor quando considerado apenas os operadores Essenciais do que quando considerando todos os operadores da Proteum. Isso aconteceu porque apesar do número de mutantes gerados ter reduzido significativamente do conjunto Essencial em relação ao conjunto com todos os operadores, a quantidade de mutantes sobreviventes não foi reduzida na mesma proporção.

A próxima etapa do experimento, foi avaliar o escore de mutação obtido pelo conjunto de CT adequado ao paradigma Procedimental nos mutantes gerados no paradigma OO. Inicialmente foi verificado o escore de mutação obtido pelo conjunto de CT adequado ao critério AM considerando todos os operadores de C contra os mutantes gerados no paradigma OO. Novamente houve a necessidade de conversão dos CT, desta vez os CT implementados em C foram convertidos para Java.

No programa Árvore Binária, os 12 mutantes gerados no paradigma OO foram testados contra o conjunto de 19 casos de teste efetivos para o paradigma Procedimental obtendo um escore de 0,909 restando apenas 1 mutante vivo. O mesmo procedimento foi feito considerando apenas os operadores Essenciais no paradigma Procedimental. Desse modo, o conjunto com os 19 CT efetivos adequados ao critério AM procedimental considerando apenas os operadores Essenciais foram executados contra os 12 mutantes do paradigma OO resultando novamente no escore de mutação de 0,909.

Para o programa Árvore Binária, os resultados obtidos pelos casos de teste adequados ao paradigma Procedimental foram melhores do que no paradigma OO. Foi identificado também que o conjunto de operadores Essenciais obteve o mesmo escore obtido pelo conjunto com todos os operadores. Uma das razões desses valores serem tão próximos se deve ao fato de que a maioria dos mutantes que exigiram a construção de novos casos de teste terem sido gerados a partir de operadores contidos no conjunto Essencial (Barbosa, 1998; Barbosa *et al.*, 2001).

4.4 Considerações Finais

Neste capítulo foi detalhado todo o processo de execução do experimento, desde a preparação do ambiente até e a aplicação do critério Análise de Mutantes. A descrição dos passos executados no experimento é importante, pois permite a outros pesquisadores analisarem com mais

detalhes o estudo e replicarem o experimento com uma maior facilidade. No Capítulo 5 serão apresentados os resultados após a execução dos testes em todos os programas.

CAPÍTULO

5

Análise dos Resultados e Avaliação da Hipóteses

5.1 Considerações Iniciais

Neste capítulo são apresentados os resultados obtidos após a execução de todos os testes e avaliações em todos os programas. Primeiramente, são mostrados os resultados referentes a cada paradigma, para que então estes dados possam ser comparados em relação ao mesmo programa implementado no outro paradigma de interesse.

Também são descritos os passos realizados para a avaliação das hipóteses do experimento, os métodos estatísticos utilizados e os resultados da avaliação de cada hipótese definida no Capítulo 3. Alguns dos resultados apresentados neste capítulo foram publicados em Campanha *et al.* (2010) em que foi avaliado o custo e o *strength* do critério considerando o paradigma OO e o paradigma Procedimental.

5.2 Resultados no Paradigma Procedimental

Os testes dos programas implementados no paradigma Procedimental foram os que gastaram mais tempo para serem aplicados. Além dos programas terem sido testados considerando dois

conjuntos de operadores diferentes, em geral, foram gerados mais mutantes do que no paradigma OO o que exigiu mais esforço para analisar cada mutantes sobrevivente.

Nas Tabelas 5.1 e 5.2 são apresentados os resultados da aplicação do critério AM a partir do conjunto de CT adequado à técnica Funcional considerando o conjunto de operadores Essenciais e o conjunto com todos os operadores da Proteum. Além da identificação do programa a tabela mostra a quantidade de mutantes gerados e a quantidade de mutantes equivalentes identificados. A partir da quarta coluna, são apresentados os dados referentes a aplicação do conjunto de casos de teste inicial: a quantidade de casos de teste, a quantidade de mutantes vivos após a execução desses casos de teste e o escore de mutação obtido. Como nem todos os conjuntos de CT obtiveram o escore de 100%, é apresentada também a quantidade de CT adicionais para que o critério fosse satisfeito, o total de CT e quantos deles são efetivos.

Tabela 5.1: Métricas referentes a execução do critério AM considerando os operadores Essenciais a partir do conjunto adequado à técnica Funcional.

Programa	Mutantes	M. Equiv.	CT Ini.	Mut. Vivos	Escore	CT Adc.	Total CT	CT. Efet.
1	96	12	3	13	0,845	2	5	5
2	152	19	5	21	0,842	6	11	11
3	152	21	5	20	0,847	7	12	12
4	483	49	5	152	0,650	19	24	24
5	184	30	4	8	0,948	1	5	4
6	98	7	3	0	1,000	0	3	2
7	56	8	3	0	1,000	0	3	2
8	292	48	7	51	0,791	10	17	16
9	556	89	7	8	0,983	4	11	7
10	546	75	5	24	0,949	3	8	5
11	192	38	14	64	0,584	3	17	9
12	36	4	7	2	0,938	1	8	7
13	68	14	17	1	0,981	1	18	9
14	38	7	10	3	0,903	2	12	9
15	174	28	14	14	0,904	3	17	8
16	45	5	10	2	0,950	1	11	9
17	996	226	25	36	0,953	29	38	22
18	493	101	21	48	0,878	5	26	17
19	1184	239	25	37	0,961	10	35	28
20	318	45	12	73	0,733	11	21	18
21	196	15	14	44	0,757	11	25	19
22	317	55	11	35	0,866	7	18	12
23	439	63	11	35	0,907	7	18	14
24	671	88	19	88	0,849	10	30	25
25	466	82	18	67	0,826	10	27	23
26	838	145	19	135	0,805	9	27	20
27	362	57	1	56	0,816	5	6	6
28	537	96	5	12	0,973	4	9	7
29	875	29	4	174	0,794	6	10	8
30	899	192	3	117	0,835	9	12	11
31	1041	185	32	36	0,958	10	32	27
32	576	85	10	118	0,760	11	21	17
Média	**418,00**	**67,41**	**10,91**	**46,84**	**0,868**	**6,78**	**16,75**	**12,91**

Tabela 5.2: Métricas referentes a execução do critério AM considerando todos os operadores de C a partir do conjunto adequado à técnica Funcional.

Programa	Mutantes	M. Equiv.	CT Ini.	Mut. Vivos	Escore	CT Adc.	Total CT	CT. Efet.
1	192	16	3	14	0,920	2	5	5
2	290	23	5	24	0,910	6	11	11
3	290	24	5	24	0,910	7	12	12
4	946	56	5	266	0,701	19	24	24
5	492	37	4	8	0,982	2	6	5
6	116	7	3	0	1,000	0	3	2
7	150	8	3	5	1,000	0	3	3
8	999	55	7	81	0,914	11	18	17
9	1549	115	7	12	0,992	5	12	10
10	1329	89	5	37	0,970	3	8	6
11	460	51	14	179	0,562	3	17	10
12	83	10	7	2	0,973	1	8	7
13	171	24	17	3	0,980	1	18	10
14	120	20	10	6	0,940	2	12	9
15	461	31	14	44	0,898	3	17	8
16	112	12	10	2	0,980	1	11	9
17	2388	299	25	60	0,971	15	40	24
18	1172	132	21	91	0,913	5	26	21
19	2676	285	25	76	0,968	14	36	34
20	657	59	12	141	0,764	11	24	17
21	440	30	14	55	0,866	11	25	19
22	799	101	11	104	0,851	7	18	12
23	1260	145	11	101	0,909	7	18	14
24	2248	155	19	122	0,942	11	30	25
25	1458	143	18	173	0,868	11	29	23
26	2504	256	19	361	0,839	9	27	20
27	1128	113	1	191	0,812	6	7	7
28	1491	158	5	49	0,963	5	10	8
29	2810	305	4	174	0,931	11	15	13
30	3183	287	3	464	0,840	10	13	11
31	2533	338	32	47	0,979	9	41	30
32	1588	176	10	302	0,786	12	22	19
Média	**1127,97**	**111,25**	**10,91**	**100,56**	**0,901**	**6,88**	**17,69**	**13,91**

Ao observar as duas tabelas pode-se verificar que o custo da aplicação do critério AM mostrou-se menor quando considerado o conjunto de operadores Essenciais, principalmente em relação à quantidade de mutantes gerados e a quantidade de mutantes equivalentes identificados. Além disso o escore de mutação obtido pelos conjuntos de CT adequados à técnica Funcional foi maior no conjunto com todos os operadores de C. Isso pode ter acontecido devido a maior dificuldade em matar os mutantes gerados pelo conjunto de operadores Essenciais em relação ao conjunto com todos os operadores da Proteum.

As Tabelas 5.3 e 5.4 apresentam a mesma avaliação, considerando como conjunto de CT inicial o conjunto adequado às técnicas Funcional e Estrutural:

Tabela 5.3: Métricas referentes a execução do critério AM considerando os operadores Essenciais a partir do conjunto adequado às técnicas Funcional e Estrutural.

Programa	Mutantes	M. Equiv.	CT Ini.	Mut. Vivos	Escore	CT Adc.	Total CT	CT. Efet.
1	96	12	3	13	0,845	2	5	5
2	152	19	5	21	0,842	6	11	11
3	152	21	5	20	0,847	7	12	12
4	483	49	8	86	0,802	18	26	26
5	184	30	4	6	0,961	1	5	4
6	98	7	3	0	1,000	0	3	2
7	56	8	3	0	1,000	0	3	2
8	292	48	7	51	0,791	10	17	16
9	556	89	8	8	0,983	4	12	7
10	546	75	5	24	0,949	3	8	5
11	192	38	14	64	0,584	3	17	9
12	36	4	7	2	0,938	1	8	7
13	68	14	17	1	0,981	1	18	9
14	38	7	10	3	0,903	2	12	9
15	174	28	14	14	0,904	3	17	8
16	45	5	10	2	0,950	1	11	9
17	996	226	30	25	0,968	9	38	22
18	493	101	27	42	0,893	5	32	17
19	1184	239	33	33	0,965	9	42	27
20	318	45	15	43	0,842	11	26	18
21	196	15	15	34	0,812	10	25	19
22	317	55	14	23	0,912	7	21	14
23	439	63	14	17	0,955	5	19	15
24	671	88	20	88	0,849	10	31	23
25	466	82	33	49	0,872	6	39	25
26	838	145	25	51	0,926	8	33	28
27	362	57	2	53	0,826	5	7	7
28	537	96	6	12	0,973	4	10	8
29	875	29	6	20	0,976	5	11	8
30	899	192	5	117	0,835	9	14	11
31	1041	185	43	11	0,987	6	49	28
32	576	85	15	91	0,815	9	24	19
Média	**418,00**	**67,41**	**13,32**	**32,00**	**0,896**	**5,63**	**18,94**	**13,44**

Ainda no paradigma Procedimental, foi verificado o *strength* do critério AM considerando o conjunto de todos os operadores da Proteum em relação ao conjunto de operadores Essencial. Esses resultados são apresentados na Tabela 5.5. Além da identificação do programa é mostrada a quantidade de mutantes gerados pelo conjunto com todos os operadores da Proteum, a quantidade de CT efetivos para o conjunto de operadores Essenciais, a quantidade de mutantes vivos após a aplicação desses CT contra os mutantes gerados e o escore de mutação obtido.

Tabela 5.4: Métricas referentes a execução do critério AM considerando todos os operadores de C a partir do conjunto adequado às técnicas Funcional e Estrutural.

Programa	Mutantes	M. Equiv.	CT Ini.	Mut. Vivos	Escore	CT Adc.	Total CT	CT. Efet.
1	192	16	3	14	0,920	2	5	5
2	290	23	5	24	0,910	6	11	11
3	290	24	5	22	0,917	7	12	12
4	946	56	8	109	0,878	18	26	26
5	492	37	4	8	0,982	2	6	5
6	116	7	3	0	1,000	0	3	2
7	150	8	3	0	1,000	0	3	3
8	999	55	7	81	0,914	11	18	17
9	1549	115	8	12	0,992	5	13	10
10	1329	89	5	37	0,970	3	8	6
11	460	51	14	179	0,562	3	17	10
12	83	10	7	2	0,973	1	8	7
13	171	24	17	3	0,980	1	18	10
14	120	20	10	6	0,940	2	12	9
15	461	31	14	44	0,898	3	17	8
16	112	12	10	2	0,980	1	11	9
17	2388	299	30	40	0,981	10	40	25
18	1172	132	27	74	0,929	5	32	21
19	2676	285	33	71	0,970	13	46	33
20	657	59	15	73	0,878	11	26	19
21	440	30	15	38	0,907	10	25	19
22	799	101	14	56	0,920	7	21	14
23	1260	145	14	57	0,949	5	19	15
24	2248	155	20	122	0,942	11	31	25
25	1458	143	33	142	0,892	7	40	28
26	2504	256	25	127	0,944	8	33	28
27	1128	113	2	186	0,817	6	8	8
28	1491	158	6	49	0,963	5	11	8
29	2810	305	6	163	0,935	10	16	13
30	3183	287	5	464	0,840	10	15	12
31	2533	338	43	15	0,993	5	48	33
32	1588	176	15	230	0,837	11	26	24
Média	**1127,97**	**111,25**	**13,31**	**76,56**	**0,922**	**6,22**	**19,53**	**14,84**

Os resultados mostram que os operadores Essenciais obtiveram um escore bastante alto em relação ao conjunto de todos os operadores da Proteum. É importante observar que apesar da quantidade de mutantes gerados e a quantidade de mutantes equivalentes terem sido reduzidas significativamente, a quantidade de casos de teste não seguiu a mesma proporção, sendo muito próximas entre os dois conjuntos de operadores.

Tabela 5.5: Escore de mutação obtido pelo conjunto de CT adequados aos operadores Essenciais de C nos mutantes gerados por todos os operadores da Proteum

Programas	Mutantes	CT	Mut. Vivos	Escore
1	192	5	0	1,000
2	290	11	0	1,000
3	290	12	0	1,000
4	946	26	0	1,000
5	492	4	1	0,998
6	116	2	0	1,000
7	150	2	3	0,979
8	999	16	3	0,997
9	1549	7	1	0,999
10	1329	5	0	1,000
11	460	9	1	0,998
12	83	7	0	1,000
13	171	9	0	1,000
14	120	9	0	1,000
15	461	8	0	1,000
16	112	9	0	1,000
17	2388	22	2	0,999
18	1172	17	1	0,999
19	2676	28	54	0,977
20	657	18	0	1,000
21	440	19	0	1,000
22	799	14	0	1,000
23	1260	15	0	1,000
24	2248	23	9	0,996
25	1458	25	2	0,998
26	2504	28	0	1,000
27	1128	7	7	0,993
28	1491	8	0	1,000
29	2810	8	104	0,958
30	3183	11	5	0,998
31	2533	28	1	1,000
32	1588	19	1	0,999
Média	**1127,97**	**13,46**	**6,10**	**0,997**

5.3 Resultados no Paradigma OO

Como o número de mutantes gerados nos programas implementados no paradigma OO foi em geral menor que no mesmo programa no paradigma Procedimental o tempo gasto para aplicação do critério AM foi menor neste paradigma. No entanto a ferramenta MuClipse apresentou alguns problemas de performance sendo que em alguns programas o tempo de execução dos conjuntos de CT contra os mutantes gerados passava de uma hora.

Na Tabela 5.6 são apresentados os resultados da aplicação do critério AM no paradigma OO considerando como conjunto de CT inicial o conjunto adequado à técnica Funcional. Em seguida na Tabela 5.7, a mesma avaliação é apresentada considerando o conjunto adequado às técnicas Funcional e Estrutural como conjunto inicial.

Tabela 5.6: Métricas referentes a execução do critério AM no paradigma OO a partir do conjunto adequado à técnica Funcional.

Programa	Mutantes	M. Equiv.	CT Ini.	Mut. Vivos	Escore	CT Adc.	Total CT	CT. Efet.
1	24	1	3	1	0,957	1	4	2
2	63	7	5	0	1,000	0	5	4
3	63	7	5	1	0,982	1	6	5
4	191	26	5	40	0,885	5	10	10
5	67	5	4	0	1,000	0	3	1
6	51	6	3	0	1,000	0	3	2
7	51	6	3	0	1,000	0	3	2
8	151	29	7	25	0,975	1	8	7
9	225	11	7	0	1,000	0	7	1
10	238	5	5	0	1,000	0	5	3
11	68	8	14	8	0,867	3	17	9
12	22	6	7	0	1,000	0	7	2
13	34	3	17	2	0,935	2	19	8
14	15	1	10	7	0,500	4	14	7
15	84	9	14	3	0,960	2	16	7
16	19	1	10	3	0,833	1	11	4
17	428	45	25	6	0,984	3	28	9
18	37	2	21	3	0,914	2	23	4
19	494	35	25	0	1,000	0	25	9
20	156	11	12	20	0,862	4	18	10
21	12	1	14	2	0,818	1	15	6
22	160	17	11	4	0,972	2	13	5
23	301	25	11	0	1,000	0	11	10
24	319	27	19	17	0,969	6	25	19
25	249	45	18	46	0,907	4	23	13
26	438	53	19	116	0,930	9	28	22
27	148	34	1	43	0,912	1	2	2
28	202	44	5	56	0,646	5	10	8
29	128	16	4	1	0,991	1	4	2
30	140	7	3	10	0,925	2	5	4
31	442	5	32	0	1,000	0	42	17
32	242	20	10	0	1,000	0	16	7
Média	**164,44**	**16,19**	**10,91**	**12,78**	**0,929**	**1,88**	**13,31**	**6,91**

Tabela 5.7: Métricas referentes a execução do critério AM no paradigma OO a partir do conjunto adequado às técnicas Funcional e Estrutural.

Programa	Mutantes	M. Equiv.	CT Ini.	Mut. Vivos	Escore	CT Adc.	Total CT	CT. Efet.
1	24	1	4	1	0,957	1	5	2
2	63	7	6	0	1,000	0	6	4
3	63	7	6	1	0,982	1	7	5
4	191	26	9	9	0,945	3	12	10
5	67	5	4	0	1,000	0	4	1
6	51	6	4	0	1,000	0	4	2
7	51	6	4	0	1,000	0	4	2
8	151	29	8	3	0,975	1	9	7
9	225	11	8	0	1,000	0	8	1
10	238	5	6	0	1,000	0	6	3
11	68	8	14	8	0,867	3	17	9
12	22	6	7	0	1,000	0	7	2
13	34	3	17	3	0,903	2	19	8
14	15	1	10	7	0,500	4	14	7
15	84	9	14	3	0,960	2	16	7
16	19	1	10	3	0,833	1	11	4
17	428	45	32	0	1,000	0	32	9
18	37	2	21	3	0,914	2	23	4
19	494	35	26	0	1,000	0	26	9
20	156	11	18	7	0,952	2	20	13
21	12	1	15	2	0,818	1	16	6
22	160	17	13	4	0,972	2	15	5
23	301	25	17	0	1,000	0	17	10
24	319	27	19	9	0,969	6	25	19
25	249	45	22	17	0,917	4	26	15
26	438	53	25	27	0,930	3	28	22
27	148	34	2	10	0,912	1	3	2
28	202	44	9	4	0,975	3	12	10
29	128	16	5	1	0,991	1	5	2
30	140	7	4	10	0,925	2	6	4
31	442	5	42	0	1,000	0	42	17
32	242	20	16	0	1,000	0	16	7
Média	**164,44**	**16,19**	**13,03**	**4,13**	**0,944**	**1,41**	**14,41**	**7,13**

Pode-se verificar que os escores de mutação obtidos pelos conjuntos adequados às técnicas Funcional e Estrutural foram relativamente altos e em geral maiores do que no paradigma Procedimental.

Analisando as duas tabelas pode-se verificar que os CT adequados aos critérios das técnicas Funcional e Estrutural obtiveram um escore muito baixo no programa 14 responsável por implementar um pilha por meio de estruturas auto-referenciaveis. Ao verificar os mutantes sobreviventes, pode-se verificar que 5 deles foram criados a partir de operadores definidos no nível de classe, sendo que quatro foram criados a partir do operador JSI que adiciona o modificador *static* na definição de variáveis. Para matar esses mutantes foram necessários construir casos

de testes que consideravam duas instâncias de objetos da classe pilha, o que não foi necessário para adequação às técnicas funcionais e estruturais.

5.4 Comparação dos Resultados nos Dois Paradigmas

Apresentados os resultados da aplicação do critério AM nos dois paradigmas esses podem ser comparados em relação ao custo e o *strength* .

Primeiramente foi analisado o *strength* do critério AM em relação aos critérios das técnicas Funcional e Estrutural. Para isso foram construídas as Tabelas 5.8 e 5.9 respectivamente. Na primeira coluna é mostrados o identificador do programa, depois para cada conjunto de operadores avaliado é apresentada a quantidade de CT do conjunto inicial, a quantidade de mutantes vivos após a execução destes casos de teste e o escore de mutação obtido por este conjunto. Para avaliar o custo associado a aplicação do teste de mutação a partir desses conjuntos de CT é mostrado também a quantidade de CT adicionais necessária para que o critério AM fosse satisfeito.

Ao analisar as duas tabelas pode-se comprovar que o critério AM no paradigma Procedimental possui maior dificuldade de satisfação do que no paradigma OO. Pode-se verificar também que apesar de pequeno houve uma melhoria entre o escore de mutação obtido pelos conjuntos de CT Funcionais+Estruturais-Adequados em relação ao conjunto de CT Funcionais-Adequados.

Feita a análise prévia do custo e do *strength* entre o critério AM em relação as técnicas Funcionais e Estruturais, o próximo passo do estudo é avaliar o custo e o *strength* do critério AM nos dois paradigmas.

A Tabela 5.10 mostra os dados referentes ao custo do teste de mutação em C e em Java considerando os três conjuntos de operadores. A primeira coluna corresponde a identificação do programa. Para cada programa são apresentadas a quantidade de mutantes gerados, a quantidade de mutantes equivalentes identificados e a quantidade de casos de teste necessária para a satisfação do critério.

Tabela 5.8: Escore de mutação obtido pelos conjuntos de CT adequados à técnica Funcional no critério AM nos dois Paradigmas.

Prog.	Todos operadores Java				Operadores Essenciais C				Todos Operadores C			
	CT	Vivos	Escore	Extra	CT	Vivos	Escore	Extra	CT	Vivos	Escore	Extra
1	3	1	0,957	1	3	13	0,845	2	3	14	0,920	2
2	5	0	1,000	0	5	21	0,842	6	5	24	0,910	6
3	5	1	0,982	1	5	20	0,847	7	5	24	0,910	7
4	5	19	0,885	5	5	152	0,650	19	5	266	0,701	19
5	4	0	1,000	0	4	8	0,948	1	4	8	0,982	2
6	3	0	1,000	0	3	0	1,000	0	3	0	1,000	0
7	3	0	1,000	0	3	0	1,000	0	3	0	1,000	0
8	7	3	0,975	1	7	51	0,791	10	7	81	0,914	11
9	7	0	1,000	0	7	8	0,983	4	7	12	0,992	5
10	5	0	1,000	0	5	24	0,949	3	5	37	0,970	3
11	14	8	0,867	3	14	64	0,584	3	14	179	0,562	3
12	7	0	1,000	0	7	2	0,938	1	7	2	0,973	1
13	17	2	0,935	2	17	1	0,981	1	17	3	0,980	1
14	10	7	0,500	4	10	3	0,903	2	10	6	0,940	2
15	14	3	0,960	2	14	14	0,904	3	14	44	0,898	3
16	10	3	0,833	1	10	2	0,950	1	10	2	0,980	1
17	25	6	0,984	3	25	36	0,953	29	25	60	0,971	15
18	21	3	0,914	2	21	48	0,878	5	21	91	0,913	5
19	25	0	1,000	0	25	37	0,961	10	25	76	0,968	14
20	12	20	0,862	4	12	73	0,733	11	12	141	0,764	11
21	14	2	0,818	1	14	44	0,757	11	14	55	0,866	11
22	11	4	0,972	2	11	35	0,866	7	11	104	0,851	7
23	11	0	1,000	0	11	35	0,907	7	11	101	0,909	7
24	19	9	0,969	6	19	88	0,849	10	19	122	0,942	11
25	18	19	0,907	4	18	67	0,826	10	18	173	0,868	11
26	19	116	0,930	9	19	135	0,805	9	19	361	0,839	9
27	1	10	0,912	1	1	56	0,816	5	1	191	0,812	6
28	5	56	0,646	5	5	12	0,973	4	5	49	0,963	5
29	4	1	0,991	1	4	174	0,794	6	4	174	0,931	11
30	3	10	0,925	2	3	117	0,835	9	3	464	0,840	10
31	32	0	1,000	0	32	36	0,958	10	32	47	0,979	9
32	10	0	1,000	0	10	118	0,760	11	10	302	0,786	12
Média	**10,91**	**9,47**	**0,929**	**1,88**	**10,91**	**46,69**	**0,868**	**6,78**	**10,91**	**100,41**	**0,901**	**6,875**

A última linha da tabela contém a média de cada uma dessas variáveis. Pode-se observar que o teste de mutação para programas no paradigma Procedimental apresentou valores superiores em todas as variáveis de custo estudadas quando comparadas com os programas em Java, mesmo quando considerado apenas o conjunto Essencial de operadores.

Para a avaliação do *strength* do critério AM, primeiramente foi analisado o escore de mutação obtido pelos conjuntos de CT adequados ao paradigma OO no paradigma Procedimental considerando os dois conjunto de operadores. Esses resultados são apresentados na Tabela 5.11. Além da identificação de cada programa é mostrada a cardinalidade do conjunto de CT Adequados ao critério AM no paradigma OO. Para cada conjunto de operadores do paradigma Procedimental são apresentados a quantidade de mutantes vivos e o escore de mutação obtido.

Tabela 5.9: Escore de mutação obtido pelos conjuntos de CT adequados às técnicas Funcional+Estrutural no critério AM nos dois Paradigmas.

Prog.	Todos operadores Java				Operadores Essenciais C				Todos Operadores C			
	CT	Vivos	Escore	Extra	CT	Vivos	Escore	Extra	CT	Vivos	Escore	Extra
1	4	1	0,957	1	3	13	0,845	2	3	14	0,920	2
2	6	0	1,000	0	5	21	0,842	6	5	24	0,910	6
3	6	1	0,982	1	5	20	0,847	7	5	22	0,917	7
4	9	9	0,945	3	8	86	0,802	18	8	109	0,878	18
5	4	0	1,000	0	4	6	0,961	1	4	8	0,982	2
6	4	0	1,000	0	3	0	1,000	0	3	0	1,000	0
7	4	0	1,000	0	3	0	1,000	0	3	0	1,000	0
8	8	3	0,975	1	7	51	0,791	10	7	81	0,914	11
9	8	0	1,000	0	8	8	0,983	4	8	12	0,992	5
10	6	0	1,000	0	5	24	0,949	3	5	37	0,970	3
11	14	8	0,867	3	14	64	0,584	3	14	179	0,562	3
12	7	0	1,000	0	7	2	0,938	1	7	2	0,973	1
13	17	3	0,903	2	17	1	0,981	1	17	3	0,980	1
14	10	7	0,500	4	10	3	0,903	2	10	6	0,940	2
15	14	3	0,960	2	14	14	0,904	3	14	44	0,898	3
16	10	3	0,833	1	10	2	0,950	1	10	2	0,980	1
17	32	0	1,000	0	30	25	0,968	9	30	40	0,981	10
18	21	3	0,914	2	27	42	0,893	5	27	74	0,929	5
19	26	0	1,000	0	33	33	0,965	9	33	71	0,970	13
20	18	7	0,952	2	15	43	0,842	11	15	73	0,878	11
21	15	2	0,818	1	15	34	0,812	10	15	38	0,907	10
22	13	4	0,972	2	14	23	0,912	7	14	56	0,920	7
23	17	0	1,000	0	14	17	0,955	5	14	57	0,949	5
24	19	9	0,969	6	20	88	0,849	10	20	122	0,942	11
25	22	17	0,917	4	33	49	0,872	6	33	142	0,892	7
26	25	27	0,930	3	25	51	0,926	8	25	127	0,944	8
27	2	10	0,912	1	2	53	0,826	5	2	186	0,817	6
28	9	4	0,975	3	6	12	0,973	4	6	49	0,963	5
29	5	1	0,991	1	6	20	0,976	5	6	163	0,935	10
30	4	10	0,925	2	5	117	0,835	9	5	464	0,840	10
31	42	0	1,000	0	43	11	0,987	6	43	15	0,993	5
32	16	0	1,000	0	15	91	0,815	9	15	230	0,837	11
Média	**13,03**	**4,13**	**0,944**	**1,41**	**13,31**	**32,00**	**0,896**	**5,63**	**13,31**	**76,57**	**0,922**	**6,22**

Os resultados mostram que o escore obtido pelos conjuntos de CT adequados ao critério AM no paradigma OO foram relativamente baixos, principalmente quando comparados com os escores obtidos pelos conjuntos de CT adequados às técnicas Funcional e Estrutural. Assim como aconteceu no programa Árvore Binária, pode-se observar que o escore de mutação considerando os operadores Essenciais é menor do que quando considerado todos os operadores da Proteum o que reforça a suposição de que os mutantes gerados pelos operadores Essenciais

Tabela 5.10: Métricas referentes ao custo do teste de mutação nos dois paradigmas

Programas	Operadores Java			Todos operadores C			Operadores Essenciais C		
	Mutantes	M. Equiv.	CT	Mutantes	M. Equiv.	CT	Mutantes	M. Equiv.	CT
1	24	1	2	192	16	5	96	12	5
2	63	7	4	290	23	11	152	19	11
3	63	7	5	290	24	12	152	21	12
4	191	26	10	946	56	26	483	49	26
5	67	5	1	492	37	5	184	30	4
6	51	6	2	116	7	2	98	7	2
7	51	6	2	150	8	3	56	8	2
8	151	29	7	999	55	17	292	48	16
9	225	11	1	1549	115	10	556	89	7
10	238	5	3	1329	89	6	546	75	5
11	68	8	9	460	51	10	192	38	9
12	22	6	2	83	10	7	36	4	7
13	34	3	8	171	24	10	68	14	9
14	15	1	7	120	20	9	38	7	9
15	84	9	7	461	31	8	174	28	8
16	19	1	4	112	12	9	45	5	9
17	428	45	9	2388	299	25	996	226	22
18	37	2	4	1172	132	21	493	101	17
19	494	35	9	2676	285	33	1184	239	27
20	156	13	13	657	59	19	318	45	18
21	12	1	6	440	30	19	196	15	19
22	160	17	5	799	101	14	317	55	14
23	301	25	10	1260	145	15	439	63	15
24	319	27	19	2248	155	25	671	88	23
25	249	45	15	1458	143	28	466	82	25
26	438	53	22	2504	256	28	838	145	28
27	148	34	2	1128	113	8	362	57	7
28	202	44	10	1491	158	8	537	96	8
29	128	16	2	2810	305	13	875	29	8
30	140	7	4	3183	287	12	899	192	11
31	442	5	17	2533	338	33	1041	185	28
32	242	20	7	1588	176	24	576	85	19
Média	**164,44**	**16,25**	**7,13**	**1127,97**	**111,25**	**14,84**	**418**	**67,41**	**13,44**

são em média mais difíceis de matar do que os mutantes gerados pelo conjunto com todos os operadores.

Por último, na Tabela 5.12 são apresentados os escores de mutação obtidos pelos conjuntos de CT adequados ao paradigma Procedimental no paradigma OO.

Na primeira coluna é mostrado o identificador do programa seguido da quantidade de mutantes gerados pelos operadores da ferramenta OO. Para cada conjunto de operadores do paradigma Procedimental é mostrada a quantidade de casos de teste, a quantidade de mutantes sobreviventes após a execução destes conjuntos de CT contra os mutantes gerados no paradigma OO e o escore de mutação obtido.

A tabela mostra que não houve muita diferença entre o escore obtido pelos dois conjuntos de operadores do paradigma Procedimental. Além disso os escores foram, em geral, maiores do que os obtidos pelos conjuntos de CT adequados ao paradigma OO.

Tabela 5.11: Escore de mutação obtido pelos conjuntos de CT adequados ao paradigma OO no paradigma Procedimental.

Programas	CT OO	Operadores Essenciais			Todos Operadores		
		Mut.	M. vivos	Escore	Mut.	M. vivos	Escore
1	2	96	22	0,738	192	30	0,830
2	4	152	23	0,827	290	25	0,906
3	5	152	23	0,824	290	26	0,902
4	10	483	69	0,841	946	87	0,902
5	1	184	8	0,948	492	10	0,978
6	2	98	0	1,000	116	0	1,000
7	2	56	2	0,958	150	2	0,986
8	7	292	47	0,807	999	74	0,922
9	1	556	17	0,964	1549	30	0,979
10	3	546	24	0,949	1329	40	0,968
11	9	192	66	0,571	460	182	0,555
12	2	36	5	0,844	83	8	0,890
13	8	68	3	0,944	171	8	0,946
14	7	38	8	0,742	120	19	0,810
15	7	174	9	0,938	461	31	0,928
16	4	45	13	0,675	112	34	0,660
17	9	996	29	0,962	2388	49	0,977
18	4	493	76	0,806	1172	147	0,859
19	9	1184	111	0,883	2676	175	0,927
20	13	318	33	0,879	657	51	0,915
21	6	196	47	0,740	440	68	0,834
22	5	317	32	0,878	799	102	0,854
23	10	439	4	0,989	1260	7	0,994
24	19	671	60	0,897	2248	140	0,933
25	15	466	86	0,776	1458	267	0,797
26	22	838	68	0,902	2504	182	0,919
27	2	362	45	0,852	1128	146	0,856
28	10	537	10	0,977	1491	47	0,965
29	2	875	28	0,967	2810	147	0,941
30	4	899	53	0,925	3183	195	0,933
31	17	1041	47	0,945	2533	86	0,961
32	7	576	97	0,802	1588	238	0,831
Média	**7,12**	**418,00**	**36,40**	**0,867**	**1127,97**	**82,906**	**0,896**

Assim como na avaliação dos critérios das técnicas funcional e estrutural, pode-se observar que o programa 14 obteve um escore de mutação mais baixo que os demais. O motivo desta diferença foi por causa dos operadores definidos para identificar características específicas do paradigma OO. Novamente a necessidade de construir casos de testes considerando mais de uma instancia da estrutura pilha é exigida no paradigma OO enquanto que não é necessária no paradigma Procedimental.

Como os resultados mostram que o critério AM no paradigma OO apresentou as medidas de custo menores do que no paradigma Procedimental e também um baixo *strength* foi feita uma análise dos mutantes sobreviventes ao conjunto de CT adequados ao paradigma OO.

Para isso os mutantes do paradigma Procedimental que sobreviveram aos conjuntos de CT adequados ao paradigma OO foram analisados e os operadores de mutação que os geraram iden-

Tabela 5.12: Escore de mutação obtido pelos conjuntos de CT adequados ao paradigma Procedimental no paradigma OO.

Programas	Mutantes OO	CT-EssenciaisC			CT-TodosC		
		CT	M. vivos	Escore	CT	M. vivos	Escore
1	24	5	0	1,000	5	0	1,000
2	63	11	0	1,000	11	0	1,000
3	63	12	0	1,000	12	0	1,000
4	191	26	12	0,927	26	12	0,927
5	67	4	0	1,000	5	0	1,000
6	51	2	0	1,000	2	0	1,000
7	51	2	0	1,000	3	0	1,000
8	151	16	0	1,000	17	0	1,000
9	225	7	0	1,000	10	0	1,000
10	238	5	0	1,000	6	0	1,000
11	68	9	6	0,900	10	6	0,900
12	22	7	2	0,875	7	2	0,875
13	34	9	5	0,839	10	5	0,839
14	15	9	4	0,714	9	4	0,714
15	84	8	2	0,973	8	2	0,973
16	19	9	2	0,889	9	2	0,889
17	428	22	0	1,000	25	0	1,000
18	37	17	2	0,943	21	2	0,943
19	494	27	2	0,996	33	2	0,996
20	156	18	4	0,972	19	2	0,986
21	12	19	1	0,909	19	1	0,909
22	160	14	6	0,958	14	6	0,958
23	301	15	0	1,000	15	0	1,000
24	319	23	7	0,976	25	7	0,976
25	249	25	19	0,907	28	19	0,907
26	438	28	0	1,000	28	0	1,000
27	148	7	3	0,974	8	3	0,974
28	202	8	7	0,956	8	7	0,956
29	128	8	0	1,000	13	0	1,000
30	140	11	5	0,962	12	5	0,962
31	442	28	0	1,000	33	0	1,000
32	242	19	0	1,000	24	0	1,000
Média	**164,43**	**13,43**	**2,78**	**0,958**	**14,84**	**2,71**	**0,959**

tificados. O resultado desta análise é apresentado na Tabela 5.13. A tabela mostra a identificação do operador de mutação, a quantidade de mutantes vivos considerando todos os programas e a porcentagem de mutantes restantes para aquele operador. Essa tabela foi construída considerando apenas os operadores Essenciais do paradigma Procedimental. A descrição do conjunto dos operadores da ferramenta Proteum pode ser obtido em Delamaro e Maldonado (2001).

Ao analisar a Tabela 5.13 pode-se identificar diversos operadores que poderiam ser acrescentados a ferramenta MuClipse para aumentar o *strength* do critério AM no paradigma OO quando comparado com o Procedimental. Dentre eles, destacam-se o operador VDTR que verifica os valores positivo, negativo e zero de cada variável e o operador VTWD que troca uma referência escalar pelo seu valor antecessor e sucessor. Esse operador funciona de forma semelhante

Tabela 5.13: Relação dos operadores que deixaram mutantes vivos dos programas em C após a execução do conjunto de CT adequados critério AM em Java

Operadores	Qtd. mutantes vivos	%
vdtr	430	36,94
vtwd	220	18,90
ccsr	137	11,77
ssdl	85	7,30
cccr	79	6,79
orrn	74	6,36
oeba	72	6,19
oasn	31	2,66
smtc	14	1,20
olbn	11	0,95
swdd	11	0,95
Total	**1164**	**100,00**

ao AOIS implementado na MuClipse, com a diferença de que a alteração do valor da variável não se propaga até o fim do algoritmo, mas somente na linha em que houve a mutação. De acordo com Vincenzi. (2004), dentre os operadores listados, apenas o operador olbn não pode ser aplicado a linguagem Java.

É importante ressaltar que não foi avaliada a capacidade desses operadores em encontrar defeitos em programas no paradigma OO, portanto estudos complementares são necessários para determinar se os operadores listados na Tabela 5.13 são representativos para a linguagem Java.

Após a execução do experimento e da organização dos dados coletados, as hipóteses devem ser avaliadas. Essa etapa é apresentada em detalhes na próxima seção.

5.5 Avaliação das Hipóteses

A análise preliminar dos resultados mostra que todas as três medidas de custo (quantidade de mutantes gerados, quantidade de mutantes equivalentes e quantidade de casos de teste) foram em geral, maiores para o teste de mutação nos programas implementados no paradigma Procedimental. Pode-se observar também que o critério apresentou maior *strength* no paradigma procedimental em relação ao Orientado a Objetos. Para avaliar as hipóteses do experimento é necessária a execução de testes estatísticos que aceitem ou rejeitem as hipóteses descritas.

Hipótese 1

A primeira hipótese diz respeito ao custo da atividade de teste. A hipótese nula afirma que não existe diferença no custo do teste de mutação nos dois paradigmas. Para avaliar tal hipótese dividimos o custo em três componentes:

- *Hipótese Nula* $H1a.0$*:* Não existe diferença de "quantidade de mutantes gerados" (QMG) em razão do paradigma, entre o conjunto de programas A (Procedimental) e B (OO).

  $H1a.0$: QMG(A) = QMG(B)

- *Hipótese Alternativa* $H1a.1$*:* Existe diferença de "quantidade de mutantes gerados" (QMG) em razão do paradigma, entre o conjunto de programas A (Procedimental) e B (OO).

  $H1a.1$: QMG(A) $\neq$ QMG(B)

- *Hipótese Nula* $H1b.0$*:* Não existe diferença de "quantidade de mutantes equivalentes identificados" (QMEI) em razão do paradigma, entre o conjunto de programas A (Procedimental) e B (OO).

  $H1b.0$: QMEI(A) = QMEI(B)

- *Hipótese Alternativa* $H1b.1$*:* Existe diferença de "quantidade de mutantes equivalentes identificados" (QMEI) em razão do paradigma, entre o conjunto de programas A (Procedimental) e B (OO).

  $H1b.1$: QMEI(A) $\neq$ QMEI(B)

- *Hipótese Nula* $H1c.0$*:* Não existe diferença de "quantidade de CT" (QCT) em razão do paradigma, entre o conjunto de programas A (Procedimental) e B (OO).

  $H1b.0$: QCT(A) = QCT(B)

- *Hipótese Alternativa* $H1c.1$*:* Existe diferença de "quantidade de CT" (QCT) em razão do paradigma, entre o conjunto de programas A (Procedimental) e B (OO).

  $H1b.1$: QCT(A) $\neq$ QCT(B)

Como o critério AM no paradigma procedimental foi avaliado considerando dois conjuntos de operadores diferentes, a avaliação deve ser feita individualmente. Os testes foram feitos primeiramente confrontando o paradigma OO com o paradigma Procedimental considerando apenas os operadores Essenciais. Para avaliar tais hipóteses, primeiramente é necessária a verificação da normalidade do conjunto de dados. Para isso foi utilizado o teste de Shapiro-Wilk

(Conover, 1998) com nível de significância $(\alpha) = 0,05$ para os três conjuntos de medidas de custo. Verificou-se que a quantidade de mutantes gerados e a quantidade de mutantes equivalentes identificados não apresentaram distribuição normal de dados. Já o conjunto da cardinalidade dos conjuntos de casos de teste adequados apresentou uma distribuição normal.

Verificada a não normalidade da distribuição das duas primeiras variáveis, elegeu-se a aplicação do teste não-paramétrico de Wilcoxon (*Signed-Rank*)(Hollander e Wolfe, 1999) com nível de significância $(\alpha) = 0,05$.

O teste que avaliou a quantidade de mutantes gerados apresentou um *p-value* = $4,657e^{-10}$, que é o menor valor representado pelo teste e muito abaixo do nível de significância assumido, que foi de 0,05. Com isso, a hipótese nula pode ser rejeitada.

Do mesmo modo, aplicou-se o mesmo teste para a avaliação da segunda hipótese. Para a quantidade de mutantes equivalentes identificado obteve-se o valor de *p-value* = $1,863e^{-09}$, também abaixo do nível de significância assumido, rejeitando-se assim a hipótese nula.

Por último utilizou-se o teste t de Student (Hollander e Wolfe, 1999) sobre a variável "quantidade de CT", uma vez que esse conjunto de dados apresentou uma distribuição normal. A hipótese nula também foi rejeitada pois no teste obteve-se um *p-value* = $5,36e^{-8}$ inferior aos 0,05 definidos como nível de significância.

Como todas as hipóteses nulas que compõem a variável custo foram rejeitadas com o nível de significância de $0,05$, pode-se concluir que existe diferença de custo entre os dois paradigmas. Com base nas médias apresentadas por essas variáveis conclui-se que o custo da aplicação do critério AM no paradigma Procedimental utilizando os operadores Essenciais da ferramenta Proteum é maior do que o custo no paradigma OO utilizando a MuClipse.

O mesmo procedimento foi realizado para comparar o custo entre os dois paradigmas considerando o conjunto com todos os operadores da Proteum:

Novamente foi verificado que a quantidade de mutantes gerados e a quantidade de mutantes equivalentes identificados não apresentavam distribuição normal de dados. Já o conjunto da cardinalidade dos conjuntos de casos de teste adequados apresentou uma distribuição normal.

O teste que avaliou a quantidade de mutantes gerados considerando todos os operadores da Proteum, também apresentou um *p-value* = $4,657e^{-10}$. Com isso, a hipótese nula pode ser rejeitada.

Para a quantidade de mutantes equivalentes identificado, ao aplicar o teste de Wilcoxon, obteve-se o *p-value* = $4,657e^{-10}$, e assim a hipótese nula foi rejeitada.

Como a variável "quantidade de CT" apresentou uma distribuição normal, foi utilizado o teste t de Student (Hollander e Wolfe, 1999). A hipótese nula também foi rejeitada pois obteve-se um *p-value* = $6,801e^{-8}$.

Como todas as hipóteses nulas que compõem a variável custo foram rejeitadas com o nível de significância de $0,05$, conclui-se também que existe diferença de custo entre os dois paradigmas. Assim, conclui-se que o custo da aplicação do critério AM no paradigma Procedimental considerando todos os operadores da Proteum também é maior do que o custo no paradigma OO utilizando a MuClipse.

Hipótese 2

Para a avaliação do *strength* do teste de mutação entre os paradigmas Procedimental e OO, o mesmo procedimento foi executado. Novamente os dados foram avaliados individualmente, começando pelo conjunto de operadores Essenciais. Primeiramente verificou-se que esses valores apresentavam uma distribuição normal por meio do teste de Shapiro-Wilk (Conover, 1998). Elegeu-se então o teste t de Student (Hollander e Wolfe, 1999) com nível de significância $(\alpha) = 0,05$ para avaliar a hipótese que foi rejeitada por apresentar um *p-value* = $2,855e^{-6}$, inferior ao nível de significância assumido.

Do mesmo modo aplicou-se o teste t de Student considerando todos os operadores da Proteum e o *p-value* obtido foi de $0,0002045$, também inferior ao nível de significância assumido rejeitando-se assim a hipótese de que o *strength* é o mesmo entre os dois paradigmas.

Com base nas médias atingidas pelos escore de mutação pode-se concluir que o *strength* do teste de mutação em C em relação a Java é significativamente maior do que no teste de mutação em Java em relação a C quando utilizadas as ferramentas Proteum e MuClipse.

Hipótese 3

Foi avaliado também o custo da aplicação dos operadores Essenciais em relação ao conjunto com todos os operadores no paradigma Procedimental. Para a avaliação do custo foram considerados mais uma vez o número de mutantes gerados, o número de mutantes equivalentes identificados e a quantidade de CT necessárias para a satisfação do critério.

Desta vez, nenhuma das variáveis de custo apresentou uma distribuição normal sendo assim foi aplicado o teste de Wilcoxon com nível de significância $(\alpha) = 0,05$ para a validação das hipóteses.

A quantidade de mutantes gerados gerou um *p-value* = $4,657e^{-10}$, a quantidade de mutantes equivalentes apresentou um *p-value* = $1,863e^{-9}$ e a quantidade de CT um *p-value* de $7,63e^{-6}$. Como todos os *p-value* apresentaram valores menores que o nível de significância assumido, a hipótese de que o custo é estatisticamente igual entre os dois conjuntos de operadores para este domínio de programas pode ser rejeitada.

Analisando a Tabela 5.10 pode-se observar que o custo da aplicação do critério AM considerando todos os operadores de mutação da Proteum é maior do que o custo quando considerado apenas os operadores Essenciais.

5.6 Considerações Finais

Neste capítulo foram apresentados todos os dados coletados após a realização dos testes e das comparações de custo e *strength* . Também foram descritos como foram realizados os testes estatísticos para avaliar as hipóteses do experimento. Com esses dados os pesquisadores podem avaliar todo o processo de experimentação, desde o planejamento, até a análise e interpretação dos resultados.

Como já explicado anteriormente, os resultados obtidos possuem grande influência das ferramentas utilizadas para o apoio ao teste de mutação. É interessante que o experimento possa ser repetido com um número maior de participantes para que os resultados obtidos possam ser comparados.

CAPÍTULO

6

Conclusão

6.1 Considerações Finais

Este trabalho teve por objetivo avaliar o impacto do paradigma de programação sobre o critério de teste Análise de Mutantes. Para isso foi realizado um estudo experimental comparando o custo e *strength* do critério Análise de Mutantes aplicados a programas do paradigma Procedimental e Orientado a Objetos, no domínio de estrutura de dados. O estudo apresentado complementa o trabalho de Prado (2009) em que foram avaliados o custo e o *strength* dos critérios da técnica Estrutural nos dois paradigmas.

Também foram avaliados o *strength* do critério AM em relação aos critérios das técnicas Funcional e Estrutural nos dois paradigmas e o custo e *strength* do conjunto com todos os operadores da Proteum em relação ao conjunto dos operadores Essenciais (Barbosa, 1998; Barbosa *et al.*, 2001) no paradigma Procedimental. Foi observado que tanto o custo quanto o *strength* possuem forte dependência dos operadores de mutação implementados em cada ferramenta. Neste sentido, foram listados alguns operadores de mutação da Proteum que, caso sejam significativos para a linguagem Java, poderiam ser implementados nas ferramentas MuJava e MuClipse para aumentar o *strength* do critério AM no paradigma OO em relação ao paradigma Procedimental.

6.2 Contribuições

Pode-se destacar como principais contribuições deste trabalho:

1. Os resultados sobre a comparação de custo e *strength* do critério Análise de Mutantes em programas no paradigma Procedimental e OO no domínio de estrutura de dados.

2. Os resultados sobre a comparação de custo e *strength* do conjunto de operadores Essenciais em relação ao conjunto com todos os operadores da ferramenta Proteum no domínio de estrutura de dados.

3. Os resultados sobre a comparação de *strength* das técnicas Funcional e Estrutural em relação ao critério Análise de Mutantes, no domínio de programas de estrutura de dados nos paradigmas Procedimental e OO.

4. A identificação de operadores de Mutação que poderiam ser adicionados a ferramenta MuClipse/MuJava para aumentar o *strength* do conjunto de operadores da ferramenta em relação ao paradigma Procedimental.

5. A geração de um conjunto de artefatos contendo a documentação de toda atividade de teste executada que além de documentar as atividades executadas no experimento podem servir como material que auxilie o ensino e treinamento das técnicas de testes de software.

6.3 Dificuldades e Limitações

O presente trabalho apresentou algumas dificuldades e limitações durante a preparação e execução:

A primeira dificuldade encontrada foi durante a escolha das ferramentas de testes, pois a quantidade de ferramentas disponíveis para a execução do critério Análise de Mutantes é bastante limitada. Além disso, muitas dessas ferramentas são bastante instáveis e possuem um conjunto limitado de operadores de mutação.

Outra dificuldade encontrada foi a falta de padronização dos operadores de mutação definidos. Apesar de existirem diversos estudos avaliando diversos conjuntos de operadores em diferentes linguagens de programação, não existe um padrão entre as ferramentas de testes. Desse modo, não é possível avaliar o critério sem levar em consideração a ferramenta de apoio utilizada. Espera-se que os artefatos gerados neste estudo possam servir para a avaliação e comparação de novas ferramentas que venham a ser construídas para o apoio ao teste de mutação.

Também foi identificada como uma limitação do experimento a quantidade de indivíduos participantes. Apesar dos casos de teste terem sido construídos seguindo sistematicamente o teste de mutação, muitas vezes foi necessária a construção de casos de teste com o objetivo de diferenciar o mesmo mutante, o que pode ter causado uma semelhança entre os casos de teste construídos nas diferentes perspectivas. Com a replicação do experimento com uma quantidade maior de participantes os resultados obtidos neste estudo podem ser comparados e o impacto do fator humano pode ser melhor avaliado.

6.4 Trabalhos Futuros

Como trabalhos futuros decorrentes desta dissertação destacam-se: a replicação desse estudo, considerando outros domínios de programas, outras linguagens de programação e um número maior de participantes envolvidos.

Também deve ser considerada a replicação do experimento considerando diferentes ferramentas de teste, já que os operadores existentes em cada ferramenta podem influenciar significativamente no custo e no *strength* gerando resultados diferentes dos obtidos neste estudo.

O conjunto de artefatos gerados por este experimento pode também ser utilizados para avaliar outras características não consideradas neste experimento, como por exemplo a porcentagem de mutantes equivalentes gerados por cada operador de mutação e a identificação de mutantes que poderiam ter sua equivalência identificada de forma automática.

Aproveitar as soluções construídas para aperfeiçoar a ferramenta Proteum de modo que a mesma aceite casos de testes implementados de forma semelhante a JUnit. Implementar as soluções utilizadas para adequação dos programas a Proteum diretamente na ferramenta, para que

Referências Bibliográficas

AGRAWAL, H.; ALBERI, J. L.; HORGAN, J. R.; LI, J. J.; LONDON, S.; WONG, W. E.; GHOSH, S.; WILDE, N. Mining system tests to aid software maintenance. v. 31, n. 7, p. 64–73, 1998.

AMARAL, E. A. G. G.; TRAVASSOS, G. H. A package model for software engineering experiments. *ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2003)*, 2003.

AMMANN, P.; OFFUTT, J.; XU., W. Coverage computation web applications. Online, 2010 `http://cs.gmu.edu:8080/offutt/coverage/`, last access January 2010., 2008.

BARBOSA, E. F. *Uma contribuição para a determinação de um conjunto essencial de operadores de mutação no teste de programas C*. Dissertação de Mestrado, ICMC/USP, São Carlos, SP, 1998.

BARBOSA, E. F.; CHAIM, M. L.; VINCENZI, A. M.; DELAMARO, M. E.; JINO, M.; MALDONADO, J. C. Introdução ao teste de software. 1st ed, cap. Teste Estrutural, Campus / Elsevier, 2007.

BARBOSA, E. F.; MALDONADO, J. C.; VINCENZI, A. M. R. Toward the determination of sufficient mutant operators for C. *Software Testing, Verification and Reliability Journal*, v. 11, n. 2, p. 113–136, john Wiley & Sons, 2001.

BARBOSA, E. F.; VINCENZI, A. M. R.; MALDONADO, J. C. Uma contribuição para determinação de um conjunto essencial de operadores de mutação no teste de programas C. In: *XII Simpósio Brasileiro de Engenharia de Software*, Maringá, PR, 1998, p. 103–120.

BASILI, V.; ZELKOWITZ, M.; MCGARRY, F.; PAGE, J.; WALIGORA, S.; PAJERSKI, R. Sel's software process improvement program. *IEEE Softw.*, v. 12, n. 6, p. 83–87, 1995.

BASILI, V. R. The experimental paradigm in software engineering. In: *Proceedings of the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions*, London, UK: Springer-Verlag, 1993, p. 3–12.

BERTOLINO, A. The (im)maturity level of software testing. *WERST Proceedings/ACM SIGSOFT*, v. 29, n. 05, p. 1–4, 2004.

BINDER, R. V. Modal testing strategies for OO software. *Computer*, v. 29, n. 11, p. 97–99, 1996.

BINDER, R. V. *Testing object-oriented systems: Models, patterns, and tools*, v. 1. Addison Wesley Longman, Inc., 1999.

CAMPANHA, D.; SOUZA, S.; MALDONADO, J. Mutation testing in procedural and object-oriented paradigms: An evaluation of data structure programs. *XXIV Simpósio Brasileiro de Engenharia de Software*, , n. XXIV, p. 91–100, 2010.

CHAIM, M. L. *Poke-Tool — uma ferramenta para suporte ao teste estrutural de programas baseados em análise de fluxo de dados*. Dissertação de Mestrado, DCA/FEEC/UNICAMP, Campinas, SP, 1991.

CHOI, B. J.; DEMILLO, R. A.; KRAUSER, E. W.; MATHUR, A. P.; MARTIN, R. J.; OFFUTT, A. J.; PAN, H.; SPAFFORD, E. H. The mothra toolset. In: *Twenty-Second Annual Hawaii International Conference on System Sciences*, HI, 1989.

CONOVER, W. J. *Practical nonparametric statistics*. John Wiley & Sons, 1998.

DELAMARO, M. E. *Proteum – um ambiente de teste baseado na análise de mutantes*. Dissertação de Mestrado, ICMC/USP, São Carlos, SP, 1993.

DELAMARO, M. E.; BARBOSA, E. F.; VINCENZI, A. M.; MALDONADO, J. C. Introdução ao teste de software. 1st ed, cap. Teste de Mutação, Campus / Elsevier, 2007a.

DELAMARO, M. E.; MALDONADO, J. C. Mutation testing for the new century. cap. Proteum/IM 2.0: An Integrated Mutation Testing Environment, Norwell, MA, USA: Kluwer Academic Publishers, p. 91–101, 2001.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. Introdução ao teste de software. 1st ed, cap. Conceitos Básicos, Campus / Elsevier, 2007b.

DEMILLO, R. A. *Software testing and evaluation*. The Benjamim/Commings Publishing Company, Inc, 1978.

DEMILLO, R. A. Mutation analysis as a tool for software quality assurance. In: *COMPASAQ80*, 1980.

DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. Hints on test data selection: Help for the practicing programmer. v. 11, n. 4, p. 34–41, 1978.

ELER, M. M. *Um serviço para o teste estrutural e certificação de serviços web e componentes de software*. Qualificação de doutorado, Instituto de Ciências Matemáticas e de Computação - ICMC-USP, 2008.

FABBRI, S. C.; VINCENZI, A.; MALDONADO, J. C. Introdução ao teste de software. 1st ed, cap. Teste Funcional, Campus / Elsevier, 2007.

FABBRI, S. C. P. F.; MALDONADO, J. C.; DELAMARO, M. E.; MASIERO, P. C. Proteum/FSM : Uma ferramenta para apoiar a validação de máquinas de estado finito pelo critério análise de mutantes. In: *IX Simpósio Brasileiro de Engenharia de Software*, Recife, PE, 1995, p. 475–478.

FONSECA, R. P. *Suporte ao teste estrutural de programas FORTRAN no ambiente POKE-TOOL*. Dissertação de Mestrado, DCA/FEE/UNICAMP, Campinas, SP, 1993.

FRANKL, P.; WEYUKER, E. A data flow testing tool. In: *Proceedings of IEEE Softfair II Conference on Software Development Tools, Techniques, and Alternatives*, 1985, p. 46–53.

FRANKL, P. G.; WEISS, S. N.; HU, C. All-uses versus mutation testing: An experimental comparison of effectiveness. *The Journal of Systems and Software*, v. 38, p. 235–253, 1996.

HARROLD, M. J. Testing: A roadmap. In: *22th International Conference on Software Engineering*, 2000.

HARROLD, M. J.; ROTHERMEL, G. Performing data flow testing on classes. ACM Press, 1994, p. 154–163.

HOLLANDER, M.; WOLFE, D. A. *Nonparametric statistical methods, 2nd edition*. Wiley-Interscience, 1999.

IEEE *IEEE standard glossary of Software Engineering terminology*. Padrão 620.12, IEEE, 1990.

IRVINE, S.; PAVLINIC, T.; TRIGG, L.; CLEARY, J.; INGLIS, S.AND UTTING, M. Jumble Java byte code to measure the effectiveness of unit tests. In: *Testing: Academic and Industrial Conference Practice and Research Techniques*, 2007, p. 169–175.

LEITÃO, P. S. J. *Suporte ao teste estrutural de programas cobol no ambiente POKE-TOOL*. Dissertação de Mestrado, DCA/FEE/UNICAMP, Campinas, SP, 1992.

LEMOS, O. A. L. *Uma contribuição para o teste estrutural de programas orientados a aspectos*. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação - ICMC-USP, 2009.

LI, N.; PRAPHAMONTRIPONG, U.; OFFUTT, J. An experimental comparison of four unit test criteria: Mutation, edge-pair, all-uses and prime path coverage. In: *Software Testing, Verification and Validation Workshops, 2009. ICSTW '09. International Conference on*, 2009, p. 220 –229.

LINKMAN, S.; VINCENZI, A. M. R.; MALDONADO, J. An evaluation of systematic functional testing using mutation testing. In: *7th International Conference on Empirical Assessment in Software Engineering - EASE*, Keele, UK, 2003, p. 1–15.

MALDONADO, J. C. *Critérios potenciais usos: Uma contribuição ao teste estrutural de software*. Tese de Doutorado, DCA/FEEC/UNICAMP, Campinas, SP, 1991.

MALDONADO, J. C.; DO ROCIO SENGER DE SOUZA, S.; FABBRI, S. C. P. F.; BARBOSA, E. F.; CHAIM, M. L.; VINCENZI, A. M. R.; DELAMARO, M. E.; JINO, M. Introdução ao teste de software. 1st ed, cap. Estudos Teóricos e Experimentais, Campus / Elsevier, 2007.

MALDONADO, J. C.; VINCENZI, A. M.; BARBOSA, E. F.; SOUZA, S. R.; DELAMARO, M. E. *Aspectos teóricos e empíricos de teste de cobertura de software*. Relatório Técnico, ICMC - USP, São Carlos, SP, 1998.

MALDONADO, J. C.; VINCENZI, A. M. R.; BARBOSA, E. F.; DELAMARO, M. E. *Teste estrutural e de mutação no contexto de programas OO*. Relatório Técnico 31, ICMC/USP, São Carlos, SP, notas Didáticas do ICMC, Série Computação, 2005.

MATHUR, A. P.; WONG, W. E. Evaluation of the cost of alternative mutation strategies. In: *VII Simpósio Brasileiro de Engenharia de Software*, Rio de Janeiro, RJ, Brazil, 1993, p. 320–335.

MCCABE, T. J.; H., W. A. *Structured testing: A testing methodology using the cyclomatic complexity metric*. NIST Special Publication 500-235, Computer Systems Laboratory National Institute of Standards and Technology Gaithersburg, MD. USA, 1996.

MOORE, I. JesTer-the JUnit test tester. http://jester.sourceforge.net, 2008.

MYERS, G. J.; SANDLER, C.; BADGETT, T.; THOMAS, T. M. *The art of software testing*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2004.

NETO, B. B.; SCARMINIO, I. S.; BRUNS, R. E. *Como fazer experimentos - pesquisa e desenvolvimento na ciência e na indústria*. Editora UNICAMP, 2001.

OFFUTT, J.; MA, Y. S.; KWON, Y. R. The class-level mutants of MuJava. In: *AST '06: Proceedings of the 2006 international workshop on Automation of software test*, New York, NY, USA: ACM, 2006, p. 78–84.

PRADO, M. P. *Avaliação de critérios de teste nos paradigmas procedural e OO: um estudo experimental*. Dissertação de Mestrado, ICMC/USP, 2009.

PRESSMAN, R. S. *Software engineering: A practitioner's approach*. 6th. ed. McGraw-Hill, 2005.

RAPPS, S.; WEYUKER, E. J. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, v. 11, n. 4, p. 367–375, 1985.

RIEHLE, D. Junit 3.8 documented using collaborations. *SIGSOFT Softw. Eng. Notes*, v. 33, n. 2, p. 1–28, 2008.

ROSA, A. C. A.; MARTINS, E. Using a reflexive architecture to validate object-oriented applications by fault injection. In: *Workshop on Reflexive Programming in C++ and Java*, 1988, p. 76–80.

ROUNTEV, A.; MILANOVA, A.; RYDER, B. G. Class analysis for testing of polymorphism in java software. *IEEE Transactions on Software Engineering*, v. 30, p. 210–220, 2004.

SIMÃO, A. S. *Proteum-RS/PN: Uma ferramenta para a validação de redes de petri baseada na análise de mutantes*. Dissertação de Mestrado, ICMC-USP, São Carlos, SP, 2000.

SMITH, B.; WILLIAMS, L. An empirical evaluation of the MuJava mutation operators. 2007, p. 193 –202.

SMITH, B.; WILLIAMS, L. On guiding the augmentation of an automated test suite via mutation analysis. *Empirical Software Engineering*, v. 14, p. 341–369, 10.1007/s10664-008-9083-7, 2009.
Disponível em `http://dx.doi.org/10.1007/s10664-008-9083-7`

SOMMERLAD, P.; GRAF, E. Cute: C++ unit testing easier. In: *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, New York, NY, USA: ACM, 2007, p. 783–784.

SOMMERVILLE, I. *Engenharia de software*. 6nd. ed. Addison Wesley, 2003.

SOUZA, S.; MALDONADO, J.; VERGÍLIO, S. Análise de mutantes e potenciais-usos: Uma avaliação empírica. *VIII International Conference on Software Technology - VIII CITS*, 1997.

SUGETA, T.; MALDONADO, J. C.; FABBRI, S. C. P. F. Proteum-RS/ST - uma ferramenta para apoiar a validação de especificações statecharts baseada no critério análise de mutantes. In: *Caderno de Ferramentas do XIII SBES - Simpósio Brasileiro de Engenharia de Software*, Florianópolis, SC, 1999, p. 41–44.

TRAVASSOS, G. *Introdução à engenharia de software experimental*. Relatório Técnico RT-ES 590/02, PESC COPPE/UFRJ, 2002.

VINCENZI, A. M.; DOMINGUES, A.; DELAMARO, M. E.; JINO, M.; MALDONADO, J. C. Introdução ao teste de software. 1st ed, cap. Teste Estrutural, Campus / Elsevier, 2007.

VINCENZI., A. M. R. *Orientação a objetos: Definição, implementação e análise de recursos de teste e validação*. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, ICMC-USP, São Carlos, SP, 2004.

VINCENZI, A. M. R.; WONG, W. E.; DELAMARO, M. E.; MALDONADO, J. C. JaBUTi: A coverage analysis tool for Java programs. In: *XVII SBES – Simpósio Brasileiro de Engenharia de Software*, Manaus, AM, Brasil, 2003, p. 79–84.

WEYUKER, E. J. The cost of data flow testing: an empirical study. *IEEE Transactions on Software Engineering*, v. 16, n. 2, p. 121–128, 1990.

WEYUKER, E. J.; WEISS, S. N.; HAMLET, R. G. Comparison of program testing strategies. In: *4th Symposium on Software Testing, Analysis and Verification*, Victoria, British Columbia, Canada, 1991, p. 1–10.

WOHLIN, C.; RUNESON, P.; HOST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, 2000a.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, 2000b.

WONG, W. E. *On mutation and data flow*. Tese de Doutorado, Department of Computer Science, Purdue University, W. Lafayette, IN, 1993.

WONG, W. E.; MATHUR, A. P.; MALDONADO, J. C. *Mutation versus all-uses: An empirical evaluation of cost, strength and effectiveness*, cap. 40 London: Chapmann & Hall, p. 258–265, 1995.

ZIVIANI, N. *Projeto de algoritmos com implementações em Java e C++*. Thomson, 2005a.

ZIVIANI, N. *Projeto de algoritmos com implementações em Pascal e C*. 2nd. ed. Thomson, 2005b.