Teste de Linha de Produto de Software Baseado em Mutação de Variabilidades¹

Marcos Antonio Quináia²

Johnny Maíkeo Ferreira³

Silvia Regina Vergilio³

Resumo: O OVM (*Orthogonal Variability Model*) tem sido adotado para definir e documentar as variabilidades de uma linha de produto de software (LPS), e para derivar produtos para teste. Dado o enorme espaço de variabilidades da maioria das aplicações, critérios de teste baseados em combinações de variabilidades, ou no teste de caminhos básicos, são geralmente utilizados para selecionar um subconjunto de produtos, os mais representativos. Estes critérios, entretanto, não consideram defeitos que podem estar presentes no OVM. Para aumentar a confiança de que os produtos da LPS satisfazem a especificação, neste trabalho é apresentada uma abordagem baseada em teste de mutação. Operadores de mutação são introduzidos e avaliados em dois estudos de caso. Os resultados mostram que outros tipos de defeitos são revelados em comparação com os critérios existentes.

Palavras-chave: Orthogonal Variability Model, Linha de Produto de Software.

Abstract: The Orthogonal Variability Model (OVM) has been adopted to define and document software product line variabilities and to derive products for testing. Given the huge variability space of most applications, testing criteria based either combinatorial testing of variabilities or on the basis path testing are generally used to select a subset of the most representative products. However, these criteria do not consider faults that can be present in the OVM. To increase the confidence that the LPS products match their requirements, in this work, a mutation based approach is presented. Mutation operators are introduced and evaluated in two case studies. The results show that other kind of faults are revealed in comparison with existing criteria.

Keywords: Orthogonal Variability Model, Software Product Line.

-
- 1 Este trabalho é uma extensão do artigo homônimo publicado no SAST'2012, indicado como melhor artigo do evento e convidado para submissão à RITA.
 - 2 Departamento de Ciência da Computação, UNICENTRO, Campus CEDETEG, Guarapuava-PR, CEP: 85040-080, {quináia} @ unicentro.br
 - 3 Departamento de Informática, C.P. 19081, UFPR, Centro Politécnico, Jardim das Américas, Curitiba-PR, CEP: 19031-970, {jmferreira,silvia} @ inf.ufpr.br

1 Introdução

Uma linha de produto de software (LPS) pode ser entendida como um conjunto de sistemas de software que compartilham características (*features*) comuns e que satisfazem às necessidades específicas de um segmento particular de mercado ou domínio. A utilização de LPS visa à criação sistemática e eficiente de produtos por meio do reúso de artefatos e também leva à customização em massa de produtos de software, diminuindo os custos de produção [9], [24].

Para alcançar estes objetivos, um suporte adequado ao gerenciamento e à definição de variabilidades da LPS é de fundamental importância [25]. Variabilidade é um conceito chave em LPS, relacionado ao fato de que sistemas computacionais possuem muitas características em comum, compartilhadas, mas ao mesmo tempo possuem uma parte variável, específica de cada aplicação. Neste sentido a variabilidade pode ser entendida como a parte que diferenciará um produto de software de outro [22]. Esta parte variável pode ser representada por modelos declarativos que explicitam a variabilidade, como por exemplo o Modelo de Variabilidade Ortogonal (*Orthogonal Variability Model* – OVM) [17]. O OVM tem como foco a parte variável da LPS, e permite que sua definição e documentação sejam feitas de maneira isolada da parte comum, facilitando o trabalho de gerenciamento de variabilidades.

Além de facilitar a documentação e o gerenciamento de variabilidades, o OVM tem sido utilizado por muitos autores para derivar produtos para teste [3], [4]. Um produto é dado pela combinação de características comuns e variáveis da LPS e é geralmente utilizado como dado de teste para testar a especificação das variabilidades da LPS. Idealmente, todas as combinações de variabilidades representadas no OVM deveriam ser testadas, todavia, devido à complexidade inerente das aplicações, o número provável de produtos é exponencial, e o teste exaustivo é notadamente impraticável [4]. Desta forma, uma das questões de pesquisa relacionadas ao teste de LPS diz respeito a como selecionar um subconjunto desses produtos que seja capaz de revelar a maioria dos defeitos e aumentar a confiança de que os produtos representados estejam de acordo com os requisitos especificados para a LPS [8], [15], [16], [23]. Para resolver esta questão é necessário estabelecer um critério de teste que possa guiar a seleção dos produtos e também oferecer uma medida de cobertura para avaliar a qualidade de um dado conjunto de produtos [19].

Esta questão tem sido investigada na literatura. A maioria dos trabalhos existentes utiliza o teste combinatorial para selecionar os produtos [4], [11], [14], [15], [16], [25], requerendo o teste de combinações de variantes presentes no OVM. Geralmente pares de variantes (*pairwise testing*) são utilizados para compor os produtos. Uma outra linha de investigação é proposta pelo trabalho de Cabral et al. [3] que propõe o uso do teste de caminhos básicos [13], geralmente aplicado no teste estrutural de programas. Os produtos para teste são derivados através da geração dos caminhos independentes de um grafo que representa as características variáveis do OVM. Uma limitação encontrada nas propostas mencionadas é que a seleção dos produtos não considera possíveis defeitos que podem estar presentes no OVM em teste, e que podem ter sido introduzidos durante a sua criação, ou seja,

não utilizam um critério baseado em defeitos, como por exemplo o critério Análise de Mutantes [7].

O critério Análise de Mutantes tem sido utilizado para o teste de programas. A ideia é que se um programa P, em teste, não está correto, ele difere do programa desejado apenas por desvios simples. Para P são gerados diversos programas mutantes que diferem de P apenas por uma modificação sintática simples produzida por um operador de mutação. Os operadores descrevem defeitos típicos que podem estar presentes em P e que foram introduzidos durante o desenvolvimento de software. Dados de teste para diferenciar P de seus mutantes devem ser gerados, ou seja, dados que produzam diferentes saídas para P e para os mutantes. Neste caso, os mutantes são ditos mortos. A ideia é matar todos os mutantes gerados. Ao final uma medida de avaliação do teste efetuado, chamada *escore de mutação*, é fornecida, dada pelo número de mutantes gerados e mortos. Em experimentos realizados no teste de programas, critérios de teste baseados em defeitos têm se mostrado os mais eficazes em termos do número de defeitos revelados [26]. O teste baseado em defeitos de variabilidades pode aumentar a confiança de que a LPS está de acordo com sua especificação e ser utilizado de maneira complementar às propostas existentes.

Considerando este fato, este trabalho tem como objetivo propor uma abordagem de teste de variabilidades de LPS baseada em mutação. A abordagem inclui um conjunto de operadores de mutação que descrevem erros típicos do modelo OVM que são utilizados para derivar produtos para teste. Além disso, como um critério de teste, os operadores podem ser utilizados para avaliar a qualidade de um conjunto de produtos existente, através do *escore de mutação*.

A abordagem aqui descrita foi inicialmente introduzida em [18] e utilizada manualmente em um estudo de caso que mostrou a aplicabilidade dos operadores propostos e sua eficácia em revelar defeitos. O presente trabalho dá continuidade à pesquisa realizada e estende o trabalho anterior nos seguintes pontos: i) refinamento dos operadores propostos a fim de reduzir o número de mutantes gerados e consequentemente o custo do teste de mutação. Alguns operadores foram redefinidos para não gerar mutantes facilmente mortos por qualquer caso de teste, ou redundantes em termos dos defeitos descritos; ii) introdução de um processo de teste baseado em mutação. Este processo é descrito detalhadamente e foi parcialmente automatizado, através de um suporte na linguagem Java que trabalha em cooperação com o *framework* FaMa-OVM [20] para análise dos OVMs mutantes; iii) nova avaliação da abordagem: considerando o refinamento dos operadores propostos e o processo de teste, o trabalho apresenta resultados de dois estudos de caso, nos quais a abordagem é aplicada e comparada com trabalhos relacionados: o teste *pairwise* e o teste orientado a caminhos básicos.

Este trabalho está organizado na seguinte sequência. Na Seção 2 é fornecida uma breve introdução ao OVM e uma representação formal para o mesmo que será utilizada ao longo do artigo para apresentação da proposta de teste baseado em mutação. Na Seção 3 são discutidos os principais trabalhos relacionados a esta pesquisa, os quais têm como objetivo a geração de dados de teste considerando as variabilidades da LPS. Na Seção 4 a abordagem

de teste baseada em mutação é descrita. São introduzidos os operadores de mutação, exemplos de mutantes gerados, e o processo de teste proposto. Na Seção 5 são descritos os dois estudos de caso: LPSs utilizadas, passos conduzidos, seguidos da apresentação e análise dos principais resultados obtidos. Na Seção 6 são apresentados as conclusões e possíveis desdobramentos para este trabalho.

2 O Modelo de Variabilidades

O termo variabilidade denota inconstância que pode levar à mudança(s). No contexto de LPS tais mudanças são propositais (conscientes) e não eventuais; assim a variabilidade se refere à flexibilidade que se pode obter, em termos de número de produtos, ao se combinar diferentes partes componentes para formar o conjunto de produtos. No processo de desenvolvimento de software, a variabilidade diferencia o desenvolvimento de software tradicional do desenvolvimento de LPS. No primeiro, a finalidade é o desenvolvimento de um software para fins específicos, ao passo que no último se procura desenvolver uma variedade de produtos que pertencem a uma família de produtos de software.

O OVM (*Orthogonal Variability Model*) é um modelo que representa todas as partes variáveis (variabilidades) que compõem uma LPS. Um exemplo de modelo OVM é apresentado na Figura 1.

Figura 1: OVM da LPS Mobile_Utilities, adaptado de [1]

Este modelo representa as variabilidades da LPS Mobile_Utilities (adaptado de [1]). As variabilidades desta linha de produto estão associadas a certas utilidades que podem ou não estar presentes em dispositivos móveis. No diagrama estão representadas as

variabilidades e pontos de variação da LPS, assim como as relações de dependência e restrições associadas às suas partes variáveis. Esse OVM será utilizado ao longo do trabalho para ilustrar o modelo formal adotado, os operadores e processo de teste associados à abordagem de teste de mutação proposta.

Para permitir a análise automatizada de OVMs alguns modelos de representação foram propostos. Neste trabalho, adota-se o metamodelo da Figura 2, proposto por Pohl et al. [17]. Neste metamodelo um OVM ($\circ$) é dado por uma tupla $\circ = (VP, V, A, R, d)$ tal que:

- VP é o conjunto de pontos de variação vp : formado por pontos nos quais a variação ocorrerá e que definem as características do sistema que poderão ou não estar nos produtos. Na Figura 2 estes pontos são representados por triângulos. Por exemplo, *Mobile_Uilities*, no qual se pode escolher os variantes *GPS*, *Screen* e *Media*. Além disso, cada ponto de variação pode ser: interno ou externo, tipo dado por $vpType$, onde $vpType \in \{int, ext\}$. Um ponto de variação interno está associado a variantes que são visíveis apenas para os desenvolvedores, mas não para os clientes. Já um ponto de variação externo está associado a variantes que são visíveis para ambos.
- V é o conjunto de variantes v que representam, cada um, uma forma de variação, ou seja a forma de realização de uma característica. As variantes são representadas na Figura 2 por retângulos, por exemplo *GPS*, *MP3*, *Camera*.
- A é o conjunto de associações que representam dependências entre os elementos do OVM. Uma associação a é da forma $(vp, aType, \{v_1, \dots, v_n\})$, onde $vp \in VP$, $aType \in \{man, opt, alt[min, max]\}$ e $\{v_1, \dots, v_n\} \subset V$; tal que $n=1$ para $aType \in \{man, opt\}$. Cada vp deve estar associado a pelo menos um v , e cada v deve estar associado a pelo menos um vp . Com relação aos tipos de associação, têm-se:
 - i) *man*: mandatória (linha sólida na Figura 1), representa um tipo de associação obrigatória caso o ponto de variação seja instanciado; o que torna o variante obrigatório. Por exemplo a associação entre *MobileUtilities* e a variante *Screen*;
 - ii) *opt*: opcional (linha tracejada), representa um tipo de associação não obrigatória caso o ponto de variação seja instanciado, por exemplo a associação entre *MobileUtilities* e a variante *GPS*; e
 - iii) *alt[min, max]*: alternativa (linha tracejada), associação entre um ponto de variação e dois ou mais variantes que exige dois números, *min* e *max*, representando respectivamente a quantidade mínima e máxima de variantes presentes caso o ponto de variação seja instanciado, com valores *default* *min* = *max* = 1. Por exemplo no ponto de variação *ScreenOptions*, apenas 1 variante deve ser selecionado entre *Basic*, *Colour* e

High_Resolution. Já o ponto de variação MediaOptions permite que 1 ou 2 variantes sejam selecionados, entre Câmera e MP3.

- R é o conjunto de restrições (linha tracejada e orientada na Figura 1) que podem ocorrer de variante para variante (v_v), de ponto de variação para ponto de variação (vp_vp), e de variante para ponto de variação (v_vp). Uma restrição r entre x e y , é dada pela tupla $(x, rType, y)$, onde $x \neq y$; $rType \in \{req, exc\}$; e, req e exc correspondem respectivamente às restrições *requires* e *excludes* indicando respectivamente que a escolha de x requer (ou exclui) y . No exemplo da Figura 1 sempre que a variante Camera for selecionada High Resolution também deve ser (restrição *requires*). Por outro lado, sempre que GPS for selecionada Basic não poderá ser (restrição *excludes*).
- $d \in VP$ representa o domínio sendo modelado pelo OVM. No caso do exemplo da Figura 1, o domínio de dispositivos móveis.

Figura 2: Metamodelo do OVM adaptado de [17]

3 Trabalhos Relacionados

O teste de LPS é um tema emergente de pesquisa e tem sido alvo na literatura de mapeamentos sistemáticos [8], [12], [23]. Estes mapeamentos mencionam diversos aspectos que podem ser enfatizados durante o teste de LPS. Dentre estes são apresentados a seguir os trabalhos voltados ao teste de variabilidades, que estão mais relacionados ao objetivo da abordagem aqui sendo proposta. Dado que o número de combinações possíveis para uma LPS pode ser enorme e o teste exaustivo de todas as combinações é impraticável, estes trabalhos têm como objetivo propor maneiras de derivar produtos para o teste de LPS a fim de cobrir as principais variabilidades e suas dependências. Um produto é gerado instanciando os pontos de variação do OVM. Um exemplo de produto, dado de teste, para o OVM da Figura 1 é descrito por (MobileUtilities; GPS; Screen; ScreenOptions; Colour).

A maior parte dos trabalhos voltados especificamente ao teste de variabilidades e à seleção de produtos para teste está baseada no teste de combinações presentes tanto em modelos de características (FM – *feature model*) como em OVM. Dentre estes, destaca-se o trabalho de McGregor [14] que utiliza métodos de teste combinatorial (*orthogonal arrays*), para cobrir o espaço de variabilidades da LPS. Cohen et al. [4] propuseram um trabalho que tem como base o OVM, *pairwise testing* e *arrays* de cobertura para selecionar os produtos. Oster et al. [15] também utilizam teste combinatorial (*pairwise*) mas diferentemente a proposta está baseada em modelos. Os trabalhos de Perrouin et al. [16] e de Uzuncaova et al. [25] têm por base o FM e fórmulas *Alloy* que são utilizadas na geração de casos de teste. Perrouin et al. [16] se valem de duas estratégias de divisão e composição para lidar com aspectos de escalabilidade no *t-wise testing*. Lamanha e Usaola [11] propõem uma extensão do AETG [5], um algoritmo tradicionalmente utilizado para realizar o teste combinatorial, para o contexto de LPS a fim de considerar restrições do tipo *requires* e *excludes*. Este algoritmo está disponível na ferramenta CombTestWeb⁴.

Diferentemente das abordagens mencionadas acima, o trabalho de Cabral et al. [3] não está baseado em teste combinatorial. A proposta chamada *FIG Basis Path* gera a partir do OVM em teste, um grafo de dependências de características (*features*) e requer a geração de produtos de tal modo que os caminhos básicos deste grafo sejam exercitados. Esta abordagem é similar ao teste de caminhos independentes, aplicado ao grafo de fluxo de controle de programas.

Observa-se que os trabalhos existentes visam a selecionar produtos para cobrir as variabilidades e suas dependências. Entretanto, não é possível garantir que o subconjunto de produtos selecionado seja capaz de revelar defeitos típicos que podem estar presentes no OVM. Isto porque os trabalhos mencionados não são baseados em defeitos. Na literatura, o trabalho de Henard et al. [10] utiliza uma abordagem baseada em defeitos, mas com um objetivo diferente. Dois operadores de mutação para fórmulas que descrevem FMs são introduzidos. Entretanto o objetivo é avaliar a similaridade de casos de teste (produtos) de

4 <http://161.67.140.42/CombTestWeb>.

um conjunto existente, e não a seleção de um conjunto dentre inúmeras possibilidades considerando defeitos presentes no OVM.

Considerando este fato, o objetivo do presente trabalho é apresentar uma abordagem de teste baseada em defeitos para derivar os produtos para teste a partir do OVM e assim aumentar a confiança de que a especificação das variabilidades esteja de acordo com os requisitos da LPS. Esta abordagem é descrita na próxima seção.

4 Abordagem de Teste Baseada em Mutação

O teste de variabilidades de LPS baseado em mutação é análogo ao teste de mutação para programas [6], [7]. Para introduzir uma abordagem baseada em mutantes é necessário: primeiramente relacionar os principais defeitos associados ao alvo em teste; propor operadores de mutação que descrevam esses defeitos para geração dos mutantes; e uma maneira de diferenciar os mutantes produzidos (execução dos mutantes) do alvo original para avaliar o conjunto de teste gerado. Esses aspectos relacionados à abordagem proposta são temas das subseções a seguir.

4.1 Operadores de Mutação

Os operadores de mutação descritos nesta seção produzem mutações no OVM original para descrever possíveis defeitos. A abordagem parte do princípio de que um OVM com defeitos define incorretamente as variabilidades da LPS, fazendo com que os produtos por ela descritos não estejam de acordo com a especificação idealizada. Os defeitos no OVM correspondem à definição incorreta de seus elementos: pontos de variação, variantes, associações e restrições.

Os operadores de mutação definidos para descrever esses defeitos são apresentados nas Tabelas 1, 2 e 3⁵. Para cada operador são fornecidas uma sigla, uma descrição e uma definição utilizando a notação introduzida na Seção 2. Dessa maneira, o OVM original é dado por $\mathbf{o} = (\mathbf{VP}, \mathbf{V}, \mathbf{A}, \mathbf{R}, \mathbf{d})$. Cada tabela corresponde a uma classe de operadores de acordo com o elemento do OVM original afetado pela mutação. Observa-se, por exemplo, que os mutantes produzidos pelos operadores da Tabela 1 têm a forma $\mathbf{m} = (\mathbf{VP}', \mathbf{V}, \mathbf{A}, \mathbf{R}, \mathbf{d})$, ou seja produzem mutações no conjunto $\mathbf{VP}$ de pontos de variação. Analogamente para os operadores das outras classes (tabelas).

5 O conteúdo das tabelas 1, 2 e 3 difere do conteúdo apresentado no trabalho anterior [18]. Os operadores, notadamente RRE e REE, foram refinados com a finalidade de reduzir o número de mutantes gerados e consequentemente o esforço de teste.

Tabela 1. Mutação em Ponto de Variação

Nome	Definição, seja $\phi = (\mathbf{VP}, \mathbf{V}, \mathbf{A}, \mathbf{R}, \mathbf{d})$	Descrição
Mutação em Ponto de variação, produz um mutante $\mathbf{m} = (\mathbf{VP}', \mathbf{V}, \mathbf{A}, \mathbf{R}, \mathbf{d})$		
MVPE	$\forall \mathbf{vp} \in \mathbf{VP} \mid \mathbf{vpType} = \text{int}$ $\mathbf{vp}' \in \mathbf{VP}' \mid \mathbf{vp}'\text{Type} = \text{ext}$	Mutação de ponto de variação interno para externo
MVPI	$\forall \mathbf{vp} \in \mathbf{VP} \mid \mathbf{vpType} = \text{ext}$ $\mathbf{vp}' \in \mathbf{VP}' \mid \mathbf{vp}'\text{Type} = \text{int}$	Mutação de ponto de variação externo para interno

Tabela 2. Mutação em Restrições

Nome	Definição, seja $\phi = (\mathbf{VP}, \mathbf{V}, \mathbf{A}, \mathbf{R}, \mathbf{d})$	Descrição
Mutação em Restrição, produz um mutante $\mathbf{m} = (\mathbf{VP}, \mathbf{V}, \mathbf{A}, \mathbf{R}', \mathbf{d})$		
R2A	$\forall \mathbf{r} \in \mathbf{R} \mid \mathbf{rType} \in \{\text{req}, \text{exc}\}$ $\mathbf{R}' = \mathbf{R} - \{\mathbf{r}\}$	Restrição Ausente
RRE	$\mathbf{r} = (\mathbf{x}, \text{req}, \mathbf{y}) \mid (\mathbf{x} \neq \mathbf{y}) \wedge (\mathbf{x}, \mathbf{y} \in \bigcup \mathbf{VP}) \wedge (\mathbf{x} \in \mathbf{VP} \square \mathbf{y} \in \mathbf{VP}) \wedge$ $(\sim \exists \mathbf{rs} = (\mathbf{x}, \mathbf{rType}, \mathbf{y}) \mid \mathbf{rs} \in \mathbf{R}) \wedge (\sim \exists \mathbf{rt} = (\mathbf{y}, \mathbf{rType}, \mathbf{x}) \mid \mathbf{rt} \in \mathbf{R})$ $\wedge (\sim \exists \mathbf{a} = (\mathbf{vp}, \mathbf{aType}, \{\mathbf{v}_1, \dots, \mathbf{v}_n\}) \mid \mathbf{x}, \mathbf{y} \in \mathbf{vpU}\{\mathbf{v}_1, \dots, \mathbf{v}_n\})$ $\mathbf{R}' = \mathbf{R} + \{\mathbf{r}\}$	Restrição Requires Extra
REE	$\mathbf{r} = (\mathbf{x}, \text{exc}, \mathbf{y}) \mid (\mathbf{x} \neq \mathbf{y}) \wedge (\mathbf{x}, \mathbf{y} \in \bigcup \mathbf{VP}) \wedge (\mathbf{x} \in \mathbf{VP} \square \mathbf{y} \in \mathbf{VP}) \wedge$ $(\sim \exists \mathbf{rs} = (\mathbf{x}, \mathbf{rType}, \mathbf{y}) \mid \mathbf{rs} \in \mathbf{R}) \wedge (\sim \exists \mathbf{rt} = (\mathbf{y}, \mathbf{rType}, \mathbf{x}) \mid \mathbf{rt} \in \mathbf{R})$ $\wedge (\sim \exists \mathbf{a} = (\mathbf{vp}, \mathbf{aType}, \{\mathbf{v}_1, \dots, \mathbf{v}_n\}) \mid \mathbf{x}, \mathbf{y} \in \mathbf{vpU}\{\mathbf{v}_1, \dots, \mathbf{v}_n\})$ $\mathbf{R}' = \mathbf{R} + \{\mathbf{r}\}$	Restrição Excludes Extra
MRR	$\forall \mathbf{r} \in \mathbf{R} \mid \mathbf{rType} = \text{exc}$ $\mathbf{r}' \in \mathbf{R}' \mid \mathbf{r}'\text{Type} = \text{req}$	Mutação de Restrição Excludes para Requires
MRE	$\forall \mathbf{r} \in \mathbf{R} \mid \mathbf{rType} = \text{req}$ $\mathbf{r}' \in \mathbf{R}' \mid \mathbf{r}'\text{Type} = \text{exc}$	Mutação de Restrição Requires para Excludes

Tabela 3. Mutação em Associações

Nome	Definição, seja $\phi = (\mathbf{VP}, \mathbf{V}, \mathbf{A}, \mathbf{R}, \mathbf{d})$	Descrição
Mutação em Associação, produz um mutante $\mathbf{m} = (\mathbf{VP}, \mathbf{V}, \mathbf{A}', \mathbf{R}, \mathbf{d})$		
RVA	$\forall \mathbf{a} \in \mathbf{A}, \mathbf{a} = (\mathbf{vp}, \text{alt}[\text{min}, \text{max}], \{\mathbf{v}_1, \dots, \mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1}, \dots, \mathbf{v}_n\}) \mid n > 2$ $\mathbf{a}' \in \mathbf{A}' \mid \mathbf{a}' = (\mathbf{vp}, \text{alt}[\text{min}, \text{max}], \{\mathbf{v}_1, \dots, \mathbf{v}_{i-1}, \mathbf{v}_{i+1}, \dots, \mathbf{v}_n\})$ $\mathbf{a}'' \in \mathbf{A}' \mid \mathbf{a}'' = (\mathbf{vp}, \text{opt}, \{\mathbf{v}_i\})$	Remove variante de uma associação alternativa (sem alterar a cardinalidade)
ILI	$\forall \mathbf{a} \in \mathbf{A}, \mathbf{a} = (\mathbf{vp}, \text{alt}[\text{min}, \text{max}], \{\mathbf{v}_1, \dots, \mathbf{v}_n\}) \mid \text{min} < \text{max} - 1$ $\mathbf{a}' \in \mathbf{A}' \mid \mathbf{a}' = (\mathbf{vp}, \text{alt}[\text{min}+1, \text{max}], \{\mathbf{v}_1, \dots, \mathbf{v}_n\})$	Incrementa limite mínimo (inferior) de uma associação alternativa

Nome	Definição, seja $\sigma = (VP, V, A, R, d)$	Descrição
DLI	$\forall a \in A, a=(vp, alt[min, max], \{v_1, \dots, v_n\}) \mid min > 0$ $a' \in A' \mid a'=(vp, alt[min-1, max], \{v_1, \dots, v_n\})$	Decrementa limite mínimo (inferior) de uma associação alternativa
ILS	$\forall a \in A, a=(vp, alt[min, max], \{v_1, \dots, v_n\}) \mid max < n$ $a' \in A' \mid a'=(vp, alt[min, max+1], \{v_1, \dots, v_n\})$	Incrementa limite máximo (superior) de uma associação alternativa
DLS	$\forall a \in A, a=(vp, alt[min, max], \{v_1, \dots, v_n\}) \mid max > min \text{ e } max > 1$ $a' \in A' \mid a'=(vp, alt[min, max-1], \{v_1, \dots, v_n\})$	Decrementa limite máximo (superior) de uma associação alternativa
MAO	$\forall a \in A \mid a=(vp, alt[min, max], \{v_1, \dots, v_n\})$ $\{a_1', \dots, a_n'\} \subset A' \mid a_1'=(vp, opt, \{v_1\}), \dots,$ $a_n'=(vp, opt, \{v_n\})$	Mutação de associação alternativa para opcional
MMO	$\forall a \in A \mid a=(vp, man, \{v\})$ $a' \in A' \mid a'=(vp, opt, \{v\})$	Mutação de associação mandatória para opcional
MAM	$\forall a \in A \mid a=(vp, alt[min, max], \{v_1, \dots, v_n\})$ $\{a_1', \dots, a_n'\} \subset A' \mid a_1'=(vp, man, \{v_1\}), \dots,$ $a_n'=(vp, man, \{v_n\})$	Mutação de associação alternativa para mandatória
MOM	$\forall a \in A \mid a=(vp, opt, \{v\})$ $a' \in A' \mid a'=(vp, man, \{v\})$	Mutação de associação opcional para mandatória
M2A	$a_1=(vp, aType, \{v_1\}), a_2=(vp, aType, \{v_2\}), \dots,$ $a_n=(vp, aType, \{v_n\}) \mid n > 1 \text{ e } aType \in \{opt, man\}$ $a' \in A' \mid a'=(vp, alt[1, n], \{v_1, \dots, v_n\})$	Mutação de associação opcional e/ou mandatória para alternativa

Para ilustrar as transformações produzidas por estes operadores, considere novamente o OVM da Figura 1 e a Figura 3. Esta figura é composta das partes a) até g), que estão dispostas em três colunas da esquerda para a direita e de cima para baixo. Cada parte mostra a aplicação de um operador de mutação sobre o OVM original, respectivamente a aplicação dos operadores MOM, DLI, RVA, MRE, RRE, MAO e MMO. Entretanto, por restrições de espaço, apenas as partes modificadas do OVM são apresentadas. A aplicação dos operadores parte do princípio de que cada mutante OVM gerado deve ser um diagrama válido, ou seja, consistente; de forma análoga aos mutantes gerados para o teste de programas que devem compilar e não conter erros sintáticos.

Figura 3: Exemplos de mutantes para o OVM da Figura 1

4.2 Processo de Teste

O processo de teste aqui descrito mostra como os operadores introduzidos na seção anterior são utilizados no teste de variabilidades descritas em um OVM em teste. Os passos desse processo estão na Figura 4, representados por um diagrama SADT™ (Structured Analysis and Design Technique) [21]. Para o teste, os operadores (todos ou alguns) podem ser selecionados. Os mutantes são então gerados (Passo 1) a partir dos operadores selecionados. Similarmente aos elementos requeridos por um critério de teste [19], os mutantes podem ser utilizados com os seguintes objetivos:

1. seleção de um conjunto P de produtos (dados de teste): nesta situação o testador não tem nenhum dado de teste disponível. Os mutantes serão utilizados para direcionar a seleção de produtos, até que um escore de mutação desejado seja obtido e se possa encerrar o teste;
2. avaliação da adequação e qualidade de um conjunto de produtos P, de acordo com o escore a ele associado. Se mais de um conjunto existir, o escore de mutação pode ser utilizado para comparar os conjuntos existentes.

Figura 4: Processo de teste utilizado

Dependendo do objetivo do teste (seleção ou avaliação) o Passo 2 pode se repetir após a execução dos mutantes e cálculo do escore. No Passo 3 os mutantes são “executados” com os casos de teste. Neste trabalho propõe-se utilizar o *framework* FaMa-OVM [20] para diferenciar os OVMs mutantes $m = (VP', V', A', R', d')$, do OVM original $o = (VP, V, A, R, d)$. O FaMa-OVM realiza a análise de diagramas OVM e é capaz de identificar se um dado produto é ou não válido para um dado diagrama. O FaMa-OVM trabalha com prova de conceitos, sendo capaz de identificar modelos vazios, elementos mortos ou falso opcionais. Esta ferramenta pode funcionar tanto *stand-alone* como integrada em um ambiente de desenvolvimento. Neste trabalho utilizou-se o ambiente de desenvolvimento Eclipse acompanhado da linguagem Java.

Os produtos de P são então validados pelo OVM original o e por seus mutantes. Se o resultado dessa validação for diferente, o mutante é considerado morto. Ou seja um mutante é morto se ele validar um produto inválido para o , ou invalidar um produto válido para o . O resultado da validação de um produto é analisado pelo testador e se for inesperado implica que o está incorreto e precisa ser corrigido.

Assim como no teste de programas, o escore de mutação é dado pelo número de mutantes mortos dividido pelo número de mutantes gerados menos os mutantes equivalentes. Um mutante é equivalente quando não existe um dado de teste capaz de diferenciar seu comportamento em relação ao comportamento do programa original, ou seja ambos computam a mesma função. Para exemplificar considere o mutante da Figura 3a. Este mutante é morto pelo dado de teste, produto, descrito por {MobileUtilities; Screen; ScreenOpt; Colour; MediaOpt; MP3} pois, este produto é válido para o diagrama original mas não é para o diagrama mutante. Já o produto {MobileUtilities; GPS; Screen; ScreenOpt; Colour; MediaOpt; MP3} não é capaz de matar este mesmo mutante, pois ele é um produto

válido para ambos diagramas. Como exemplos de mutantes equivalentes considere, os dois diagramas da Figura 5, gerados a partir da aplicação dos operadores RRE e REE. Estes mutantes são equivalentes ao original, ou seja, descrevem exatamente a mesma quantidade e o mesmo conjunto de dados de teste (produtos) que o diagrama original, não havendo caso de teste que os diferencie.

Figura 5: Exemplos de mutantes equivalentes para o OVM da Figura 1

Com o objetivo de automatizar o processo de geração de casos de teste (produtos) e a análise dos OVMs mutantes, um suporte foi desenvolvido em Java, no ambiente Eclipse, que facilita a utilização do *framework* FaMa-OVM. O programa é usado na geração dos casos de teste (produtos) e na fase de execução, na qual os mutantes são automaticamente comparados com o OVM original para a classificação dos mutantes mortos e dos mutantes equivalentes, e posterior cálculo do escore. Este programa de suporte permitiu automatizar parte do processo de teste, reduzindo esforço para aplicação dos operadores.

5 Avaliação

A avaliação da abordagem baseada em mutação foi realizada em dois estudos de caso, utilizando o OVM da LPS Mobile_Uilities, apresentado na Seção 2, e o OVM adaptado de [2] para uma LPS do domínio de automóveis, os quais serão identificados respectivamente por MU e Car. O objetivo destes dois estudos de caso é avaliar a aplicabilidade dos operadores e do processo de teste propostos, e realizar uma comparação com o teste de caminhos básicos e o teste *pairwise*. Inicialmente é dada uma descrição de como cada teste foi aplicado. Após, os resultados de cada aplicação são apresentados e avaliados.

5.1 Aplicação das abordagens de teste

As três abordagens: teste de mutação, teste *pairwise* e teste de caminhos básicos, foram aplicadas da seguinte maneira.

- *Teste de mutação*: a aplicação dos operadores propostos neste trabalho foi feita manualmente considerando-se todas as possibilidades (100%) de aplicação. Similarmente ao que acontece no teste de mutação de programas, no estudo, foram considerados somente mutantes válidos, consistentes de acordo com o *framework* FaMa-OVM. Além disso, mutantes do tipo *void*, mutantes que estão associados a um conjunto vazio de produtos válidos foram considerados anômalos. Aplicando o processo proposto, para cada OVM, foi identificado um conjunto, denominado *mutaSet*, de produtos (dados de teste) capaz de diferenciar o OVM original de seus mutantes. Caso não fosse possível encontrar tal produto o mutante foi considerado equivalente. Uma maneira automatizada de determinar a equivalência entre OVMs é possível utilizando o FaMa-OVM e checando se os produtos gerados para cada diagrama coincidem.
- *Teste de caminhos básicos*: para cada OVM foi gerado o grafo FIG [3] correspondente e um conjunto de produtos, denominado *figSet*, para cobrir seus caminhos básicos.
- *Teste pairwise*: o teste baseado em combinação [4], [5] foi realizado utilizando a ferramenta CombTestWeb, que gerou um conjunto de produtos, denominado *aetgSet*.

As abordagens foram então comparadas considerando o *strength* [13]. Os conjuntos *figSet* e *aetgSet* foram submetidos ao processo de análise de adequação de acordo com os operadores propostos e o escore associado foi obtido.

5.2 Aplicabilidade do teste de mutação

Nesta seção os resultados são utilizados para avaliar a aplicabilidade dos operadores e do processo de teste propostos tendo como base a Tabela 4, que apresenta o número de mutantes gerados e número de dados de teste requeridos por cada operador. Nesta tabela os operadores que não puderam ser aplicados receberam o acrônimo “na” (não se aplica). Observa-se que os operadores MVPE e MVPI não foram aplicados em nenhum dos dois diagramas, pois não havia em ambos diagramas nenhuma indicação com relação ao tipo de ponto de variação.

Tabela 4: Número de mutantes gerados e de produtos necessários

operador	MU				Car			
	gerados	equi-val.	vazios (anom)	#casos de teste	gerados	equi-val.	vazios (anom)	#casos de teste
MVPE	na	na	na	na	na	na	na	na
MVPI	na	na	na	na	na	na	na	na
RVA	4			3	4			3
ILI	1			1	1			1

DLI	2			2	2			2
ILS	1			1	2	1		1
DLS	1			1	1			1
MAO	2			2	2			2
MMO	1			1	na	na	na	na
MAM	2			1	2			1
MOM	2			1	1			1
M2A	1			1	na	na	na	na
R2A	4	2		2	na	na	na	na
RRE	58	27		4	32	16		2
REE	58	4	7	4	32		6	2
MRR	1			1	na	na	na	na
MRE	3		1	2	na	na	na	na
totais	141	33	8	(27)6	79	17	6	(16)3

Observa-se que os operadores MMO, M2A, R2A, MRR e MRE não puderam ser aplicados no OVM Car, devido às características intrínsecas desse diagrama. Por exemplo este diagrama não contém restrições e nem características opcionais.

Os operadores associados a restrições R2A, RRE e REE e o operador ILS para o Car foram os que geraram mutantes equivalentes, ou seja, aqueles que os produtos do mutante são exatamente iguais aos do OVM original. Dentre estes, em ambos os casos o operador RRE se destaca como sendo o operador que mais gerou equivalência.

Especificamente para o estudo de caso Mobile_Uilities, foram encontrados 33 mutantes equivalentes. Estes mutantes foram gerados pela inclusão de uma relação (*requires* ou *excludes*) entre MP3 e algum variante relacionado à escolha alternativa do ponto de variação *ScreenPoints*. Para matar estes mutantes seria necessário criar um produto que contrarie as restrições, muitas vezes incluindo o variante Camera que possui uma relação *requires* com *High_Resolution*. Isto requer que este último seja incluído, fazendo com que não seja possível incluir outros variantes Basic e Colour, devido à escolha alternativa ter cardinalidade [1,1]. Portanto qualquer produto será inválido para estes mutantes e para o OVM original, não existindo um produto capaz de diferenciar seus comportamentos.

Com relação à geração de mutantes anômalos, destaca-se a geração de diagramas OVM *void* que não estão associados a nenhum produto válido. Novamente, o operador REE gerou 7 mutantes anômalos para o MU e 6 para o Car. No caso do MU, o operador MRE gerou 1 mutante vazio.

As quinta e nona colunas da Tabela 4 apresentam o resultado da soma (entre parênteses) de todas as linhas, num total de 27 produtos (casos de teste) para o MU e 16 para o Car. Entretanto, vale a pena ressaltar que este total não corresponde ao número de casos de teste necessários. Foram necessários apenas 6 produtos diferentes para matar todos os

mutantes não equivalentes do MU e apenas 3 produtos para o Car. Em ambos os casos, os operadores RRE e REE foram os que mais geraram mutantes e consequentemente os que exigiram um maior número de casos de teste. Observa-se que a maioria dos operadores requereu um caso de teste para cada mutante gerado. Entretanto, quando o número de mutantes cresce, o número de casos de teste não cresce proporcionalmente, como é o caso dos operadores RRE e REE. Por exemplo, para o operador RRE foram necessários apenas 4 casos de teste para matar os 47 mutantes ativos do MU, e 2 para matar os 26 mutantes do Car. Isto mostra a aplicabilidade da abordagem baseada em mutação e dos operadores propostos.

Outros experimentos com outros diagramas OVM deverão ser conduzidos para que o custo possa ser melhor avaliado. Os operadores RRE e REE são os que mais geraram mutantes, como também mutantes equivalentes e mutantes vazios. Em um trabalho futuro eles deverão ser redefinidos para melhorar a aplicabilidade da proposta. Talvez um critério possa ser utilizado para se gerar ou não um mutante possuindo uma restrição extra.

5.3 Comparação das abordagens

Para a análise do *strenght* será utilizada a Tabela 5 que contém um resumo dos resultados obtidos pelos conjuntos figSet e aetgSet. Para cada operador aplicado ao diagrama tem-se o número de mutantes mortos para cada conjunto. Ao final, são apresentados o número de mutantes gerados, descontando os equivalentes e anômalos, utilizado para calcular o escore global, e o número de casos de teste de cada conjunto. Vale a pena ressaltar que para ambos diagramas todos os casos de teste dos conjuntos figSet e aetgSet foram necessários para se obter o escore apresentado, ou seja, contribuíram para matar um mutante ainda não morto pelos casos de teste executados anteriormente.

Tabela 5: Resultados obtidos pelos conjuntos figSet e aetgSet

operador	MU			Car		
	#mutaSet	#figSet	#aetgSet	#mutaSet	#figSet	#aetgSet
RVA	4	3	4	4	2	4
ILI	1	1	1	1	1	1
DLI	2	2	4	2	2	4
ILS	1	0	0	1	0	0
DLS	1	0	0	1	0	0
MAO	2	2	4	2	2	4
MMO	1	1	1	na	na	na
MAM	2	2	2	2	2	2
MOM	2	2	2	1	1	1
M2A	1	1	1	na	na	na
R2A	2	0	0	na	na	na
RRE	31	23	27	16	6	14

REE	47	31	35	26	8	22
MRR	1	1	0	na	na	na
MRE	2	2	2	na	na	na
#mutantes mortos	100	71	83	56	24	52
escore	1	0,71	0,83	1	0,43	0,93
#casos de teste	6	3	4	3	2	4

Com relação ao número de casos de teste, a abordagem de mutação é mais custosa, assim como no teste tradicional de programas. O teste combinatorial é intermediário e o teste de caminhos básicos é o que apresenta menos custo. Entretanto, este custo adicional pode ser justificado devido ao número de defeitos revelados. Observa-se que existem operadores (ILS, DLS e R2A) de mutação para os quais nenhum mutante pôde ser morto usando os casos de teste dos conjuntos figSet e aetgSet. Estes operadores estão associados aos limites máximos das associações alternativas. Isto significa que as abordagens de *pairwise* e de caminhos básicos não foram capazes de revelar os defeitos descritos por estes operadores e que o teste de mutação aqui sendo proposto deve ser usado como complementar, para aumentar a confiança de que os produtos da LPS estão de acordo com a especificação.

Os conjuntos figSet apresentaram menor escore, principalmente no caso do Car. Contrariamente, neste caso o teste combinatorial apresentou um escore melhor que para o caso MU. Uma explicação para isto é a estrutura do OVM Car que não contém muitas restrições. Percebe-se que a grande porcentagem de mutantes que não foram mortos pelos conjuntos aetgSet são relativos aos operadores de mutação em restrições RRE e REE. Uma análise visual dos conjuntos mostra que o conjunto mutaSet inclui os conjuntos figSet e aetgSet e também satisfazem ambas abordagens de teste.

6 Conclusões

Este artigo apresentou uma abordagem de teste baseada em mutação de variabilidades da LPS a partir do OVM. A ideia é utilizar um critério baseado em defeitos para a seleção de um conjunto de produtos, considerando erros comuns que podem estar presentes no OVM, descritos por um conjunto de operadores de mutação introduzido.

Aplicar operadores de mutação em diagramas OVM é essencialmente semelhante ao teste de mutação de programas. Um OVM mutante é considerado morto se o seu resultado da validação de um caso de teste (produto) é diferente do resultado produzido pelo OVM original. Ao final, um escore de mutação é obtido, e pode ser usado para guiar a geração de produtos, ou para avaliar a qualidade de um conjunto disponível.

Os operadores propostos neste trabalho foram utilizados em dois estudos de caso, nos quais os resultados mostram a aplicabilidade da abordagem. O número de casos de teste necessários não cresce proporcionalmente ao número de mutantes gerados.

A abordagem foi comparada com os testes *pairwise* e de caminhos básicos, os quais não foram capazes de matar diversos mutantes, ou seja, não revelam os defeitos descritos pelos operadores de mutação associados a estes mutantes. Os defeitos estão associados principalmente à cardinalidade das associações alternativas. Este estudo preliminar mostra que a abordagem baseada em mutação deve ser utilizada de uma forma complementar às existentes, sendo capaz de revelar outros tipos de defeitos e contribuindo para aumentar a confiança de que os produtos descritos através do OVM estão de acordo com a especificação.

Para a condução deste trabalho foi implementado um suporte automatizado para auxiliar nas etapas de geração e execução dos mutantes em cooperação com o *framework* FaMa-OVM. A etapa de geração dos mutantes neste trabalho foi realizada de forma manual, o que é uma limitação. Este processo está sujeito a erros. Isto constitui uma ameaça aos resultados aqui apresentados. No momento está sendo desenvolvido um módulo para geração automática dos mutantes, o que permitirá a condução de novos experimentos, com OVMs representativos de LPSs maiores, para uma análise mais detalhada do custo e avaliação da escalabilidade. Esta nova etapa possibilitará novos refinamentos para os operadores propostos, e/ou a definição de novos. Espera-se também, em etapa futura, a proposição de estratégias para redução do custo, da mesma forma que acontece com o teste tradicional de mutação de programas.

Agradecimentos

Os autores agradecem ao CNPq e à Fundação Araucária pelo apoio financeiro.

Referências

- [1] Benavides, D., Segura, S., Ruiz-Cortés, A. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35 (6), pp. 615–636. 2010.
- [2] Batory, D., Thaker, S. Safe Composition of Product Lines. In: *GPCE'07, 6th International Conference on Generative Programming and Component Engineering*, pp. 95-104. 2007.
- [3] Cabral, I., Cohen, M.B., Rothermel, G. Improving the testing and testability of software product lines. In: *SPLC'10, International Conference on Software Product Lines: going beyond*. Springer-Verlag, Berlin, Heidelberg. pp. 241-255. 2010.

- [4] Cohen, M.B., Dwyer, M.B., Shi, J. Coverage and adequacy in software product line testing. *In: ISSTA'06, Workshop on Role of Software Architecture for Testing and Analysis*. pp. 53-63. 2006.
- [5] Cohen, D.M., Dalal, S.R., Parelius, J., Patton, G.C. The combinatorial design approach to automatic test generation. *IEEE Software* 13(5), pp. 83-88. 1996.
- [6] Delamaro, M.E., Maldonado, J.C., Jino, M. Introdução ao Teste de Software. Série Campus – SBC, Editora Campus. 2007.
- [7] DeMillo, R. A., Lipton, R. J., e Sayward, F. G. Hints on test data selection: help for the practicing programmer. *IEEE Computer*, 11(4), pp. 34-41. 1978.
- [8] Engström, E., Runeson, P. Software product line testing: a systematic mapping study. *Information and Software Technology*. 53(1), pp. 2-13. 2011.
- [9] Greenfield, J., Short, K. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley & Sons, Indianapolis. 2004.
- [10] Henard, C., Papadakis, M., Perrouiny, G., Klein, J. e Le Traon, Y. Assessing SPL Testing via Model-based Mutation_An Application to Similarity Testing. *In: The 9th Workshop on Advances in Model Based Testing A-MOST'13*. pp. 10. 2013.
- [11] Lamancha, B.P., Usaola, M.P. Testing product generation in software product lines using pairwise for features coverage. *In: ICTSS'10. International Conference on Testing Software and Systems*. pp. 111-125. 2010.
- [12] Lamancha, B.P., Usaola, M.P., Velthius, M.P. Software product line testing, a systematic review. *In: International Conference on Software Paradigm Trends*. vol. 49, pp. 78-81. 2009.
- [13] McCabe, T. A complexity measure. *IEEE Transactions on Software Engineering*. 2(4), pp. 308–320. 1976.
- [14] McGregor, J.D. Testing a software product line. *Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-2001-TR-022*. 2001.

- [15] Oster, S., Zink, M., Lochau, M., Grechanik, M. Pairwise feature-interaction testing for SPLs: potentials and limitations. In: *SPLC'11, Proceedings of the 15th International Software Product Line Conference*. vol 2, pp. 6:1-6:8. 2011.
- [16] Perrouin, G., Sen, S., Klein, J., Baudry, B., Traon, Y.L. Automated and scalable t-wise test case generation strategies for software product lines. In: *ICST'10, Third International Conference on Software Testing, Verification and Validation*. pp. 459-468. 2010.
- [17] Pohl, K., Böckle, G. and van der Linden, F. Software Product Line Engineering Foundations, Principles, and Techniques. *Springer-Verlag, New York, USA*. 2005.
- [18] Quináia, M. A., Ferreira, J. M., Vergilio, S. R. Teste de Linha de Produto de Software Baseado em Mutação de Variabilidades. In: *SAST'12, 6th Brazilian Workshop on Systematic and Automated Software Testing*. pp.10. 2012.
- [19] Rapps, S. e Weyuker, E. J. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*. 14(4), pp. 367-375. 1985.
- [20] Roos-Frantz, F., Galindo, J. A. Benavides, D. Ruiz-Cortés, A. FaMa-OVM: a tool for the automated analysis of OVMs. In: *SPLC'12, Proceedings of the 16th International Software Product Line Conference*. pp. 250-254. 2012.
- [21] Ross, D.T. Structured Analysis (SA): A Language for Communicating Ideas. *IEEE Transactions on Software Engineering*. SE-3, pp.16-34. 1977.
- [22] SEI, Software Engineering Institute/Carnegie Mellon. Software Product Line. <http://www.sei.cmu.edu/productlines/> Acesso em dez/2012.
- [23] Silveira Neto, P.A. da M., Carmo Machado, I.d., McGregor, J.D., de Almeida, E.S., de Lemos Meira, S.R. A systematic mapping study of software product lines testing. *Information and Software Technology*. 53(5), pp. 407-423. 2011.
- [24] van der Linden, F.J., Schimd, K., Rommes, E. Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering. *Springer-Verlag, New York*. 2007.
- [25] Uzuncaova, E., Khurshid, S., Batory, D.F. Incremental test generation for software product lines. *IEEE Transactions on Software Engineering*. 36(3), pp. 309-322. 2010.

- [26] Wong, W. E., Mathur, A. P., Maldonado, J. C. Mutation versus all-uses: An empirical evaluation of cost, strength and effectiveness. *In: Lee, M.; et al.. (Org.). Software Quality and Productivity: Theory, Practice, Education and Training*. Chapman & Hall. Londres. pp. 258-265. 1995.