SPLiT-TeSGe - Um Processo para Adaptação de Métodos de Geração de Sequências de Testes para Linha de Produto de Software

Aline Zanin, Avelino Francisco Zorzo, Leandro Teodoro Costa

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - Prédio 32 - Porto Alegre - RS - Brasil

{aline.zanin, leandro.teodoro}@acad.pucrs.br, avelino.zorzo@pucrs.br

Abstract. *The software product line development has gained notoriety for being an ally to projects that seek to increase productivity through reuse of artifacts. This reuse, already used in the development process, has recently started to be adopted during the software testing phase, which is considered one of the most costly phases of the development process. In this work, we propose a process to adapt methods for generating test sequences, traditionally used in single systems, to be used in software product lines. This process is called* ***S****oftware* ***P****roduct* ***Li****ne* ***T****est using* ***T****est* ***S****equence* ***G****eneration* ***Me****thod (SPLiT_TSGe). The main idea is that test artifacts produced in the Domain Engineering are reused for products derived in the Application Engineering allowing, thus, to reduce the number of test case to test a software derived from a software product line.*

Resumo. *O desenvolvimento de software em linha de produto tem ganhado notoriedade por ser um aliado a projetos que buscam aumentar a produtividade através do reuso de artefatos. Este reaproveitamento, já utilizado no processo de desenvolvimento, recentemente passou a ser adotado também no processo de testes de software, visto que, a etapa de testes é considerada uma das etapas mais onerosas do processo de desenvolvimento. Neste trabalho buscamos propor um processo para a adaptação de métodos de geração de sequências de testes, tradicionalmente utilizados em sistemas únicos, para a utilização em Linha de Produto de Software. Este processo chama-se* ***S****oftware* ***P****roduct* ***Li****ne* ***T****esting using* ***T****est* ***S****equence* ***G****eneration* ***Me****thod (****SPLiT-TSGe****). Com isso, visamos reutilizar os artefatos de teste produzidos na Engenharia de Domínio para os produtos derivados na Engenharia de Aplicação, permitindo assim, reduzir o número de casos de teste necessários para testar produtos derivados de uma Linha de Produto de Software.*

1. Introdução

O segmento de testes vem ganhando espaço em empresas desenvolvedoras de software devido ao aumento da competitividade e exigência de qualidade nos produtos entregues aos clientes. Para atender esta demanda de produtos com qualidade, diversas técnicas de testes têm sido adotadas. Dentre estas técnicas encontra-se teste baseado em modelos (*Model Based Testing* - MBT) [Apfelbaum and Doyle 1997]. MBT consiste na utilização de modelos (formais, comportamentais e estruturais) para a geração de casos de teste.

Uma das formas de aplicação de MBT é através de **M**étodos de **G**eração de **S**equências de **T**este (MGSTs). Estes métodos recebem como entrada uma Máquina de Estados Finitos (*Finite State Machine* - FSM) [Kanjilal et al. 1995] e retornam como saída um conjunto de ações que devem ser executadas para efetuar o teste de uma aplicação.

Entretanto, embora os MGSTs sejam considerados uma forma eficiente e eficaz de realizar testes, estes métodos são comumente utilizados para testes de sistemas únicos e pouco explorados no segmento de Linha de Produto de Software (*Software Product Line*- SPL), em um momento no qual o conceito de SPL tem adquirido visibilidade em projetos de desenvolvimento que buscam aumento de produtividade. Uma SPL pode ser definida como uma família de produtos que compartilham a mesma estrutura e possuem características em comum, porém, cada produto possui peculiaridades que o diferencia dos demais produtos da SPL [(SEI) 2014].

Fazendo uso de SPL, os artefatos de desenvolvimento podem ser reutilizados reduzindo os riscos de falhas[1] e otimizando o uso de tempo e recurso [Rodrigues et al. 2014]. Este reuso de artefatos para o desenvolvimento de sistemas em SPL não é algo inovador, porém, poucos trabalhos abrangem tratam deste aspecto no processo de testes de software. Entretanto, a importância que o teste de software possui para sistemas únicos é ainda maior para uma SPL, uma vez que uma falha em um artefato reutilizável pode gerar defeitos em todos os produtos derivados.

Considerando a existência de software produzido a partir de SPLs, as vantagens da reutilização de artefatos e a importância dos testes de software, neste trabalho apresentamos um processo para adaptação de MGSTs para SPL, denominado SPLiT-TeSGe. Desta forma buscamos responder a seguinte questão de pesquisa: **como adaptar MGST para o contexto de SPL?**

Este trabalho possui a seguinte estrutura: na Seção 2 será conceituado MBT, MGSTs e SPL. Na Seção 3 será apresentado o processo SPLiT-TeSGe. Na Seção 3.1 serão apresentados os resultados da aplicação do processo proposto no método HSI. Ao final, são apresentados os trabalhos relacionados a esta pesquisa e as conclusões que obtivemos.

2. Fundamentação teórica

Nesta seção serão descritos os conceitos que fundamentam os tópicos diretamente relacionados com este trabalho, sendo estes: Teste Baseado em Modelos, Métodos de Geração de Sequências de Testes e Linha de Produto de Software.

2.1. Testes Baseados em Modelos (*Model Based Testing* - MBT)

MBT é uma técnica para a geração automática de artefatos de teste com base em modelos gerados a partir de artefatos de software. Estes modelos são desenvolvidos no início do ciclo de desenvolvimento de software, possibilitando que os testes sejam feitos de acordo com o que software deve fazer, e não em o que o software faz [Apfelbaum and Doyle 1997]. Desta forma, pode-se garantir uma maior eficácia no processo de testes. Esta técnica permite também reduzir custos com geração de artefatos de teste e correção de falhas. Além disto, permite criar uma rastreabilidade entre modelo

[1]Nesse trabalho, utilizamos os conceitos de falha, erro e defeito segundo [Avizienis et al. 2004].

e requisito, de forma que quando ocorre uma alteração de requisito de sistema seja possível recriar os artefatos de teste de forma automatizada. Assim, o problema de alto custo gerado pela manutenção destes artefatos será mitigado.

Para realização de testes com base em modelos podem ser utilizados modelos comportamentais, arquiteturais ou formais. Dentre os modelos formais comumente utilizados encontram-se as FSMs. FSMs são compostas por um conjunto de estados e transições em que transições são utilizadas para conectar estados. FSMs são uma alternativa para projetar componentes de teste de software, uma vez que são aplicáveis a qualquer modelo de especificação [Chow 1978]. Uma das formas de utilização de FSMs em MBT é através de MGSTs.

2.2. Métodos de Geração de Sequências de Teste (MGSTs)

MGSTs são mecanismos para geração de artefatos de teste para sistemas descritos a partir de modelos. Estes métodos recebem como entrada uma FSM e retornam como saída um conjunto de ações que devem ser executadas para efetuar o teste de uma aplicação, verificando assim se a implementação de um sistema está correta em relação a sua especificação. São conhecidos diversos MGSTs, por exemplo: W [Chow 1978], Wp [Fujiwara et al. 1991], Unique Input/Output (UIO) [Sabnani and Dahbura 1988] e Harmonized State Identification (HSI) [Luo et al. 1995]. Em geral, todos os métodos tem o mesmo objetivo, o que difere um do outro são as características das FSMs aplicáveis, o tamanho e a cobertura das sequências que são geradas por cada MGST. Desta forma, é preciso analisar a relação custo benefício para cada um dos métodos na definição do método adequado para cada cenário. Estes MGSTs são comumente utilizados no conceito de sistemas únicos, no entanto, por meio de uma adaptação podem ser aplicados também a SPLs.

2.3. Linha de Produto de Software (*Software Product Line* - SPL)

Conforme mencionado anteriormente, uma SPL pode ser definida como uma família de softwares similares, desenvolvidos a partir de um conjunto de especificações de requisitos comuns a todos esses sistemas, tendo a mesma base e um alto grau de reaproveitamento [(SEI) 2014]. Com a adoção de SPL é possível atingir otimização de tempo e recursos que se dá pela construção de um novo software a partir da gerência dos aspectos relativos a variabilidade entre os produtos e utilização das partes reutilizáveis de um software já existente.

Uma das principais características de uma SPL é a presença de variabilidade, *i.e.* a forma como os membros de uma linha de produto podem se diferenciar entre si [Weiss and Lai 1999]. A variabilidade é representada por pontos de variação e variantes. Um ponto de variação é um local específico de um artefato em que uma descrição do projeto ainda não foi resolvida. Para cada ponto de variação, uma ou mais variantes podem estar associadas, sendo uma variante uma característica que é escolhida para resolver um ponto de variação.

Segundo [Kang et al. 1990], uma característica pode ser opcional (*optional*), obrigatória (*common/mandatory*) ou alternativa (*alternative/Xor* e *alternative/Or*). Sendo que uma característica opcional pode ou não estar presente no produto derivado de uma SPL. Uma característica obrigatória, por sua vez, necessariamente fará parte de todos os produtos. Já características do tipo alternativa "*Xor*", são características excludentes, ou seja,

a seleção de uma característica alternativa "*Xor*" implica na exclusão das demais características alternativas pertencentes ao mesmo grupo e nível hierárquico. Existem ainda características do tipo alternativa "*Or*", onde uma ou mais características pertencentes ao mesmo grupo e nível hierárquico podem estar presentes no produto. As características também possuem relação entre si e algumas restrições são determinadas, tais como: relação de dependência (*depends/requires*) e relação de exclusão (*mutex/excludes*).

Uma SPL possui três atividades principais em sua estrutura, sendo elas [(SEI) 2014]: Engenharia de Domínio, que é a atividade que define o núcleo de artefatos comum a todos os produtos de uma SPL; Engenharia de Aplicação, que é a atividade onde são derivados os produtos, reutilizando os artefatos definidos e criados na Engenharia de Domínio e explorando as peculiaridades de cada produto, através da solução de variabilidades de uma SPL; e, Gerenciamento da Linha de Produto, que ocorre durante todo ciclo de vida da SPL e é responsável pela gestão de uma SPL.

A importância do teste de software em sistemas únicos, já bastante difundida, pode também ser considerada para SPL. Portanto, considerando a existência de softwares produzidos a partir de SPLs, é necessário que os MGSTs, atualmente utilizados para teste de sistemas únicos, sejam adaptados para gerar com eficiência sequências de teste para uma SPL. Os objetivos principais desta adaptação são o reuso de artefatos de teste e a redução do esforço na geração dos artefatos de teste. Desta forma, casos de teste e *scripts* para testar funcionalidades comuns entre os produtos são gerados uma única vez.

3. SPLiT-TeSGe - Processo para Adaptação de Métodos de Geração de Sequências de Teste para Linha de Produto de Software

Neste trabalho propomos um processo para adaptação de MGSTs para SPLs denominado **S**oftware **P**roduct **Li**ne **T**esting using **T**est **S**equence **G**eneration M**e**thods (**SPLiT-TeSGe**). Este processo é parte do método **S**oftware **P**roduct **Li**ne **T**esting **M**ethod **B**ased on Sys**t**em Models (SPLiT-MBt) [Costa et al. 2014], que propõe um método para geração de casos de teste para SPLs, a partir de modelos. O resultado do processo proposto é um conjunto de sequências de teste com informação de variabilidade, as quais são geradas durante a Engenharia de Domínio da SPL para ser posteriormente resolvida na Engenharia de Aplicação.

A grande diferença entre utilização de MGSTs para testes de sistemas únicos e para testes de SPLs é que no contexto de SPL os métodos precisam estar preparados para tratar variabilidade. Neste contexto os MGSTs devem ser capazes de identificar informações de variabilidade e gerar, a partir de cada ponto de variação, sequências de testes distintas de acordo com as variantes acopladas. O processo SPLiT-TeSGe inicia levando em consideração uma FSM que já possui informações de variabilidade e teste inseridas. Estas informações são adicionadas seguindo o estabelecido no método SPLiT-MBT [Costa et al. 2014], mais especificamente através da abordagem SMarty [Oliveira et al. 2010].

Para propormos o processo deste trabalho fez-se necessário analisar os principais MGSTs, em relação a sua estrutura. Com esta visão podemos notar que, os principais MGSTs trabalham com conjuntos de sequências *e.g.* *State Cover* (Q), *Transition Cover* (P), *Characterized Set* (W), *Identification Set* (Wi), *Harmonized Identifiers Set* (HI), *Unique sequence of input and output* (UIO), *Distinction sequence* (DS), *Synchronization*

sequence (SS). Estas sequências quando concatenadas e minimizadas resultam na sequência final de cada método. Desta forma, para propor uma forma de adaptação que possa ser utilizada genericamente para diversos MGSTs é necessário mudar o comportamento dos MGSTs no momento da geração destas sequências. E é exatamente neste momento a maior contribuição do SPLiT-TeSGe, fazendo com que as sequências geradas já possuam informações de variabilidade. Para que seja possível realizar a geração de sequência de testes com variabilidade, a partir de FSM recebida, as seguintes ações devem ser executadas:

1. O método deve reunir os estados que representam variantes pertencentes a um mesmo ponto de variação em um estado único. Este estado irá receber como nome "VP + o nome do estado anterior ao ponto de variação" e como entrada e saída, respectivamente um conjunto com as informações de entrada e um conjunto com as informações de saída de todas as variantes que pertencem aquele ponto de variação (as mesmas variantes que foram agrupadas).
2. Variantes que também representam pontos de variação são "separadas" da FSM original, formando sub-FSMs. Adotou-se esta estratégia, pois em uma FSM é possível existir um conjunto de estados vinculados a determinada variante que não seriam considerados devido a determinação do Critério 1, mas que também necessitam ser considerados para a aplicação dos métodos.
3. Tendo como base os Critérios 1 e 2, os MGSTs são aplicados em dois momentos:
   Aplica-se o método na FSM principal considerando a descrição do Critério 1;
   Aplica-se o método de geração de sequência de testes em todas as sub-FSMs, quando estas existirem.

Executadas estas ações, no momento da geração das sequências (P, Q, HI, W, WP) que são a base dos MGSTs, deve ser considerado o conjunto de entradas criado. Diferente do que acontece na abordagem tradicional, onde é considerada apenas uma entrada, as sequências de teste geradas a partir da aplicação dos MGSTs irão possuir informação de variabilidade. Isto se deve ao fato do conjunto de entradas criado fazer parte das sequências finais. A seguir, é apresentado o alfabeto que representa informações de variabilidade nas sequências (baseado em [Costa et al. 2014]):

Op = representa pontos de variação ou variantes opcionais;

VP_or = representa pontos de variação cujas variantes fazem parte de um grupo de variantes mutuamente inclusivas (`alternative_OR`);

VP_xor = representa pontos de variação cujas variantes fazem parte de um grupo de variantes mutuamente exclusivas (`alternative_XOR`);

{} = delimita o conjunto de variantes vinculados a um ponto de variação;

() = delimita o conjunto de sequências de teste gerado a partir da aplicação dos métodos na FSM;

$Req_{->}$ = representa a relação de dependência (*depends/requires*) entre variantes;

$Ex_{->}$ = representa a relação de exclusão (*excludes/mutex*) entre variantes.

Após serem geradas as sequências com variabilidade na Engenharia de Domínio, é necessário resolver esta variabilidade. A resolução é realizada utilizando a abordagem

SPLiT-MBT [Costa et al. 2014]. Com a solução da variabilidade, na Engenharia de Aplicação, são geradas sequências específicas para cada produto de uma determinada SPL. Na geração de sequências na Engenharia de Aplicação são levadas em consideração as características dos produtos e respeitadas as relações (*optional, requires, mutex*) das variabilidades de cada ponto de variação. A fim de validar este processo e facilitar a geração de sequências de testes foi realizada a implementação prática do método HSI, fazendo uso do processo SPLiT-TeSGe. A seguir é demonstrado um exemplo da aplicação do processo SPLiT-TeSGe fazendo uso do método HSI.

3.1. Aplicação do processo no método HSI

Nesta seção será descrito um exemplo de uso aplicação do processo SPLiT-TeSGe, fazendo uso do Método de Geração de Sequência de testes Harmonized State Identification - HSI. Para exemplificar a aplicação do processo utilizaremos a FSM da Figura 1. Nesta FSM podem ser visualizadas as informações de variabilidade constantes nas transições (S1-S8), (S1-S2) e (S1-S9). Neste exemplo, a variabilidade representada é do tipo ≪**alternative_OR**≫, isto é, pelo menos uma das variantes deverá ser selecionada. Desta forma, é possível derivar desta FSM sete produtos: Produto Estado S8, Produto Estado S2, Produto Estado S9, Produto Estados S2-S8, Produto Estados S2-S9, Produto Estados S8-S9, Produto Estados S2-S8-S9.

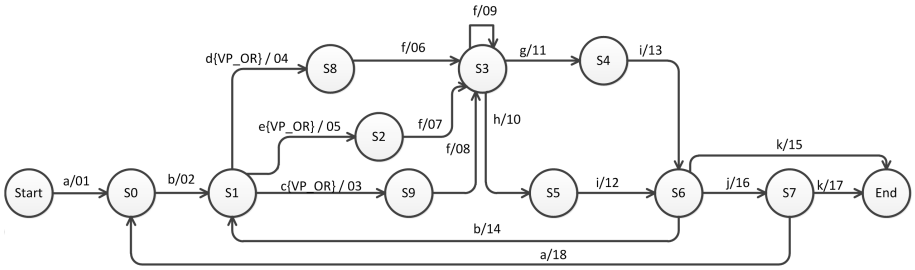

Figura 1. Máquina de Estados Finitos com variabilidade

Após receber a FSM contendo informações de variabilidade, o método deverá agrupar as variantes de um ponto de variação em um estado único, sendo que, a transição de entrada deste estado deverá conter as entradas e saídas de todas as variantes pertencentes a ele. Como resultado desta etapa, a FSM da Figura 1 será transformada na FSM da Figura 2.

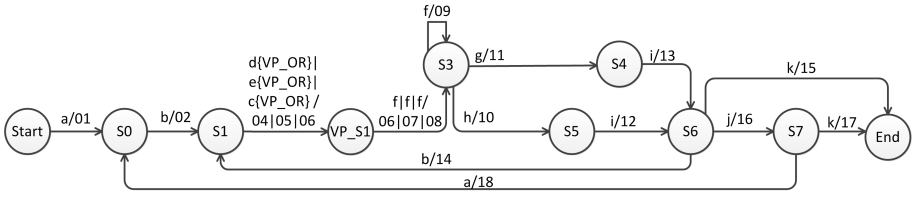

Figura 2. Máquina de Estados Finitos com ponto de variação

É possível notar que na Figura 2 a transição que antecede o estado **VP_S1** recebeu as informações de entrada e saída das transições de entrada das variantes **S8, S2 e S9**, enquanto que a transição que sucede o estado **VP_S1** recebeu as informações de entrada e saída das transições de saída das variantes citadas.

Considerando esta nova formação da FSM, as etapas do método HSI irão considerar um conjunto de entradas e não mais entradas únicas como na abordagem tradicional. A seguir, apresentam-se as etapas de geração das sequências fazendo uso do HSI. A cada etapa será demonstrado o resultado considerando o processo SPLiT-TeSGe e a abordagem tradicional.

3.2. Criação do *State Cover - Preâmbulo*

O *Preâmbulo* é determinado pelo caminho de entradas percorrido a partir estado inicial até cada estado da FSM. A Tabela 1 (a) demonstra o preâmbulo dos estados da FSM da Figura 2 utilizando o processo SPLiT-TeSGe[2].

Podemos visualizar que o preâmbulo dos estados S3, S4, S5, S7 e VP_S1 possuem variabilidade ({dec}), isto acontece porque na FSM original existiam os estados S2, S8 e S9 que possuíam respectivamente as entradas d, e, c. Como houve um agrupamento destas variantes em um estado único, o preâmbulo gerado considera este grupo de entradas, ao passo que, caso fosse testado individualmente cada produto, para cada estado posterior ao ponto de variação teríamos três preâmbulos possíveis, um para cada estado variante.

3.3. Criação do *Transition Cover*

Transition Cover é um conjunto de sequências que cobre todas as transições da FSM. Ele é obtido através da junção do caminho de entradas percorrido a partir estado inicial até cada estado (*preâmbulo*) mais as entradas das transições de saída de cada estado. A Tabela 1 (b) demonstra o *Transition Cover* dos estados da FSM da Figura 2 utilizando o processo SPLiT-TeSGe.

Estado	***State Cover***
Start	ϵ
S0	ϵ a
S1	ϵ,a,b
S2	n/a
S3	ϵ,a,b,{dec},{fff}
S4	ϵ,a,b,{dec},{fff},g
S5	ϵ,a,b,{dec},{fff},h
S6	ϵ,a,b,{dec},{fff},h,i
	ϵ,a,b,{dec},{fff},g,i
S7	ϵ,a,b,{dec},{fff},h,i,j
	ϵ,a,b,{dec},{fff},g,i,j
S8	n/a
S9	n/a
VP_S1	a,b,{dec}

Estado	***Transition Cover***
Start	ϵ a
S0	ϵ,a,b
S1	ϵ,a,b,dec
S2	n/a
S3	ϵ,a,b,dec,fff,f; ϵ,a,b,{dec},{fff},g
	ϵ,a,b,{dec},{fff},h
S4	ϵ,a,b,{dec},{fff},g,i
S5	ϵ,a,b,{dec},{fff},h,i
S6	ϵ,a,b,{dec},{fff},h,i,k; ϵ,a,b,{dec},{fff},g,i,k
	ϵ,a,b,{dec},{fff},h,i,j; ϵ,a,b,{dec},{fff},g,i,j
	ϵ,a,b,{dec},{fff},h,i,a; ϵ,a,b,{dec},{fff},g,i,a
S7	ϵ,a,b,{dec},{fff},h,i,j; ϵ,a,b,{dec},{fff},g,i,j
	ϵ,a,b,{dec},{fff},h,i,j,a; ϵ,a,b,{dec},{fff},g,i,j,k
S8	n/a
S9	n/a
VP_S1	ϵ,a,b,{dec},{fff}

Tabela 1. HSI *State Cover - (a)* HSI *Transition Cover - (b)*

Do mesmo modo que no preâmbulo, pode-se notar a presença de conjuntos de entradas, neste exemplo os conjuntos {dec} e {fff}. Note que o primeiro conjunto refere-se ao agrupamento das entradas das transições de entrada dos estados variantes pertencentes

[2]Neste exemplo a segunda variabilidade possui três entradas iguais, porém em outro exemplo estas entradas podem ser diferentes, dependendo da FSM utilizada.

a VP_S1, já o segundo conjunto refere-se ao agrupamento das entradas das transições de saída dos estados variantes pertencentes a VP_S1. Caso não fosse utilizado o processo SPLiT-TeSGe, existiria um número maior de sequências *Transition Cover*, visto que, para este exemplo, por existirem três variantes, para cada estado posterior ao ponto de variação existem três preâmbulos possíveis, que originam nove *Transition Cover*.

3.4. Geração dos Identificadores Harmonizados (HSI)

Para obter o Conjunto de Identificadores Harmonizados da FSM é criada uma tabela (veja um exemplo na Tabela 2 que apresenta um resumo dos resultados para diversos produtos), denominada de Tabela de Relação de Transitividade. Nesta tabela, são apresentadas as colunas: "Par de Estados Origem", "Par de Estados Destino", "Entradas", "Saídas" e "Resultados". A partir desta tabela, o processo para obtenção dos Identificadores Harmonizados é o que segue:

- Na coluna "Par de Estados Origem" são listados todos os pares de estado da FSM e na coluna "Entradas" são colocas as entradas pertencentes ao alfabeto de entrada da FSM, sendo que estas entradas devem ser repetidas para cada par de estados;
- Para cada par de estados (S_i,S_j), aplicam-se todas as entradas (percorrendo a FSM). Para S_i e S_j, verifica-se o estado onde a execução parou e adiciona-se este estado à coluna "Par de Estados Destino" formando um novo par (S'_i e S'_j). Caso a entrada não seja válida para alguns dos estados do par, adiciona-se à coluna "Resultados" o *status* "desconsiderado";
- Para cada par de estados (S_i,S_j), aplicam-se todas as entradas (percorrendo a FSM). Para S_i e S_j adiciona-se na coluna "Saída" a saída da transição;
- Para cada linha da tabela é verificado se as saídas da coluna "Saída" são diferentes para os estados S_i e S_j, neste caso adiciona-se à coluna "Resultados" o *status* "falho". No caso das entradas serem idênticas, adiciona-se à coluna "Resultados" o *status* válido.

Esta é a mais custosa das etapas do método HSI. A quantidade de linhas da tabela é determinada por Sp^i, sendo *Sp* o número de pares de estados da FSM e *i* o número de entradas da FSM. Por este motivo, e por serem estes os pares utilizados nas próximas etapas do HSI, optamos por apresentar neste trabalho apenas os pares de estados que obtiveram resultado falho. Nas colunas 3-8 da Tabela 2[3] é possível visualizar o resultado desta etapa quando executado o método HSI na forma tradicional, testando individualmente cada produto, sendo possível notar que diversos pares de estados retornaram como resultado falho. Na coluna 9 da Tabela 2, por sua vez, é possível visualizar o resultado desta etapa quando executado o método HSI utilizando o processo SPLiT-TeSGe. Pode-se notar que com o processo SPLIT-TeSGe foram obtidos menos resultados falhos, isto significa que o processo SPLiT-TeSGe irá reduzir o número de Identificadores Harmonizados e consequentemente de sequências geradas.

Para localizar os Identificadores Harmonizados a partir da Tabela de Relação de Transitividade, deve-se percorrer a tabela buscando a menor sequência de entradas, que para cada par de estados atinge o resultado "falho". Para isso, a entrada que gerou um *status* "válido" é concatenada com a entrada que gerou um *status* "falho" formando a

[3]A Tabela 2 representa apenas as possibilidades que possuem *status* "falho", não exibindo os *status* "desconsiderado" e "válido".

Par de Estados Origem	Entr.	Prod. S8	Prod. S9	Prod. S2	Prod. S2,S9	Prod. S2,S8	Prod. S8,S9	SPLIT TeSGe	Saída	Par de Estados Destino
Start,S7	a	**falho**	**falho**	**falho**	**falho**	**falho**	**falho**	**falho**	01, 18	S0, S0
S0,S6	b	**falho**	**falho**	**falho**	**falho**	**falho**	**falho**	**falho**	02,14	S1,S1
S8,S3	f	**falho**	n/a	n/a	n/a	**falho**	**falho**	n/a	06, 09	S3, S3
S5,S4	i	**falho**	**falho**	**falho**	**falho**	**falho**	**falho**	**falho**	12, 13	S6, S6
S6,S7	k	**falho**	**falho**	**falho**	**falho**	**falho**	**falho**	**falho**	15,17	end,end
S9,S3	f	n/a	**falho**	n/a	**falho**	n/a	n/a	n/a	08,09	S3,S3
S2,S3	f	n/a	n/a	**falho**	**falho**	**falho**	n/a	n/a	07,09	S3,S3
S2,S9	f	n/a	n/a	n/a	**falho**	n/a	n/a	n/a	07, 08	S3,S3
S8,S2	f	n/a	n/a	n/a	n/a	**falho**	n/a	n/a	06, 07	S3,S3
S8,S9	f	n/a	n/a	n/a	n/a	n/a	**falho**	n/a	06, 08	S3,S3

Tabela 2. HSI Relação de Transitividade por Produto

sequência de entradas. Alguns pares da FSM podem não ter um identificador harmonizado, por nenhuma sequência de entradas atingir um status "Falho" ou todas as entradas terem como resultado um par de estados inválido. Como resultado desta etapa podemos observar na Tabela 3 os pares com seus respectivos Identificadores Harmonizados.

Prod. S8		**Prod. S2,S9**		**Prod. S9**		**Prod. S2,S8**		**Prod. S2**		**Prod. S8,S9**		**SPLiT-TeSGe**	
St,S7	a	St,S7	a	St,S7	a	St,S7	a	St,S7	a	St,S7	a	St,S7	a
S0,S6	b	S0,S6	b	S0,S6	b	S0,S6	b	S0,S6	b	S0,S6	b	S0,S6	b
S8,S3	f	S5,S4	i	S5,S4	i	S8,S3	f	S5,S4	i	S8,S3	f	S5,S4	i
S5,S4	i	S6,S7	k	S6,S7	k	S5,S4	i	S6,S7	k	S5,S4	i	S6,S7	k
S6,S7	k	S9,S3	f	S9,S3	f	S6,S7	k	S2,S3	f	S6,S7	k		
		S2,S3	f			S9,S3	f			S9,S3	f		
		S2,S9	f			S8,S9	f			S8,S9	f		

Tabela 3. Identificador Harmonizado

Após localizar o identificador de cada par, o próximo passo é encontrar o Conjunto de Identificadores Harmonizados (HSI) da FSM, a partir dos Identificadores Harmonizados dos pares de estados. Para chegar a este resultado é necessário, para cada estado, analisar os Identificadores Harmonizados dos pares nos quais aquele estado pertence, e excluir os estados repetidos. Para exemplificar, o estado S6 pertence ao par (com identificador harmonizado), (S0,S6) que tem como identificador harmonizado "b" e ao par (S6,S7) que tem como identificador harmonizado "k", sendo assim o identificador harmonizado de S6 é "bk".

A última etapa do método HSI consiste em executar cada *Transition Cover* gerado e concatenar o valor deste com o HSI do estado em que a execução do *Transition Cover* parou. Por exemplo, para a FSM da Figura 1, o *Transition Cover* {abdecfffgi} termina sua execução do estado S6 que tem como HSI {b,k} gerando as sequências abdecfffgib e abdecfffgik. O Conjunto HSI para a FSM da Figura 2 é formado pelas sequências: ab{d;e;c}{VP_or}{fff}ff;ab{d;e;c}{VP_or}{fff}gib; ab{d;e;c}{VP_or}{fff}gik; ab{d;e;c}{VP_or}{fff}hib; ab{d;e;c}{VP_or}{fff}hik; ab{d;e;c}{VP_or}{fff}gijk; e ab{d;e;c}{VP_or}{fff}gijab;

4. Exemplo de Uso

Nesta seção é descrito como o método SPLiT-TeSGe foi aplicado para testar as funcionalidades de produtos derivados da SPL *Arcade Game Maker* (AGM) [(SEI) 2014]. A AGM foi criada pelo *Software Engineering Institute* (SEI) com o objetivo de auxiliar o aprendizado dos conceitos de SPLs por meio de uma abordagem prática. Esta SPL consiste de três diferentes jogos eletrônicos: *Bowling*, *Brickles* e *Pong*.

Para realização dos testes da AGM, efetuamos a criação de FSMs que representam as funcionalidades e as informações de variabilidade e teste de todos os produtos que podem ser derivados desta SPL. Estas FSMs representam diversas operações que podem ser executadas pelo usuário/jogador, por exemplo: instalar e desinstalar jogos (FSMs *Install Game* e *Uninstall Game*), selecionar o jogo a ser jogado (FSM *Play Selected Game*), salvar a pontuação atual de um jogador (FSM *Save Score*), verificar as melhores pontuações salvas anteriormente (FSM *Check Previous Best Score*) e finalizar um jogo que está em andamento (FSM *Exit Game*). A Figura 3 apresenta a FSM *Play Selected Game* que é uma das sete FSMs que foram criadas[4]. É importante enfatizar que nesta figura as informações de entrada e saída são apenas um conjunto de identificadores que foram utilizados para facilitar a visualização, sendo que os dados reais podem ser encontrados na página web do de nosso projeto de pesquisa[5].

A FSM *Play Selected Game* foi escolhida para demonstrar como os artefatos de teste são reutilizados para testar as funcionalidades dos produtos. Esta escolha ocorreu por ser a FSM que melhor representa as interações do usuário com os produtos da AGM. Os estados *Start*, *Standard Instances*, *Ready State*, *Initialize the game*, *Responds to Won/Lost/TiedDialog* e *Exit* representam variantes obrigatórias (`mandatory`), *i.e.*, obrigatoriamente estarão presentes em todos os produtos derivados da AGM. Por outro lado, os estados *Brickles Moves*, *Pong Moves* e *Bowling Moves* correspondem a variantes mutualmente inclusivas (`alternative_OR`) *i.e.*, os produtos derivados da AGM podem ter combinações de um, dois ou todos os três jogos. Os jogos *Brickles Moves*, *Pong Moves* e *Bowling Moves* representam funcionalidades complexas que são representadas por sub-FSMs (ver item 2 do processo SPLIT-TeSGe).

4.1. Análise

Através da aplicação do método HSI adaptado, durante a Engenharia de Domínio, foram geradas 66 sequências de teste para as sete FSMs. Essas sequências foram reutilizadas na Engenharia de Aplicação para testar as funcionalidades dos 14 produtos que foram derivados da AGM. Esses produtos representam combinações de um a três jogos, sendo o número de sequências reutilizadas por cada produto descrito na Tabela 4.

Conforme descrito na Tabela 4, foram reutilizadas 519 sequências entre estes produtos. Com base neste número, a porcentagem de reutilização das sequências de teste foi obtida através de uma métrica chamada *Size and Frequency metric* (R_{sf}) [Devanbu et al. 1996]. Considerando essa métrica, a porcentagem de reutilização de sequências de teste gerados é dada por:

[4]Todas as FSMs podem ser localizadas na seguinte página web: www.cepes.pucrs.br/AGM.

[5]A mesma página web em que estão armazenadas as FSMs: www.cepes.pucrs.br/AGM.

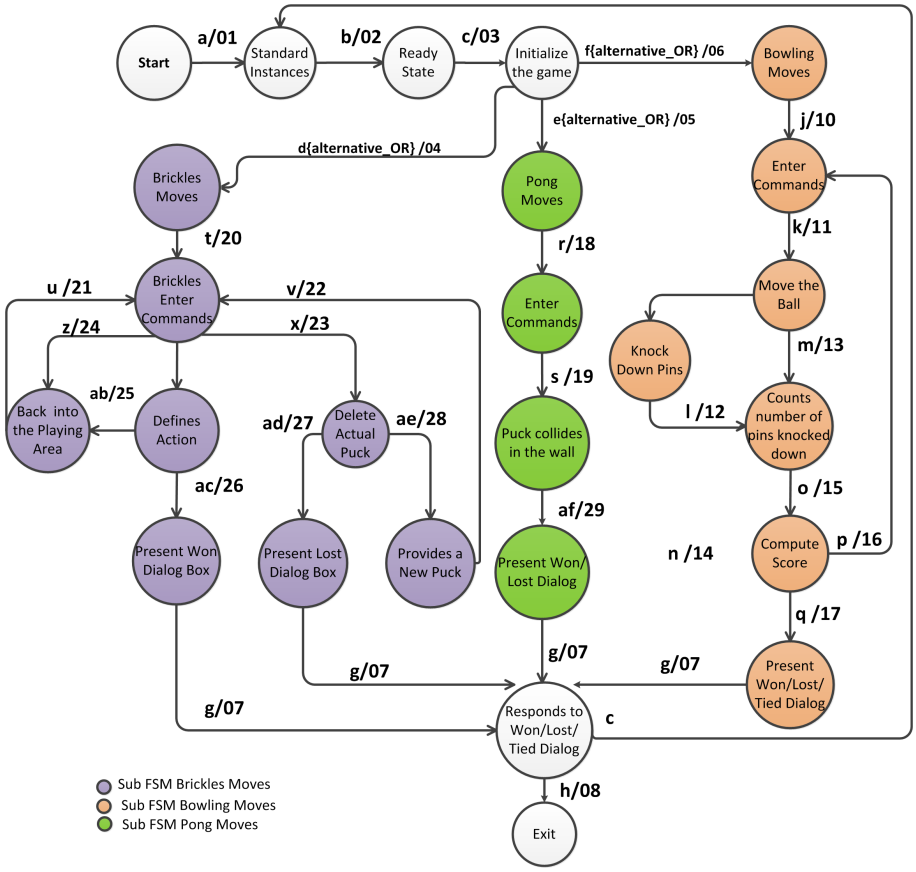

Figura 3. Finite State Machine of `Play Selected Game`

ID do Produto	Nro. de Seq. de testes	ID do Produto	Nro. de Seq. de testes	ID do Produto	Nro. de Seq. de testes
Produto 1	19	Produto 2	23	Produto 3	15
Produto 4	42	Produto 5	34	Produto 6	38
Produto 7	57	Produto 8	28	Produto 9	32
Produto 10	24	Produto 11	51	Produto 12	43
Produto 13	47	Produto 14	66		
					Total = 519

Tabela 4. Número se sequências geradas por produto

$$R_{sf} = \frac{Size_{sf} - Size_{act}}{Size_{sf}} = \frac{519 - 66}{519} = 0.87$$

Esses valores determinam que 87% das sequências de teste geradas na Engenharia de Domínio foram reutilizadas para testar funcionalidades dos 14 produtos derivados a partir da AGM. Desta forma, o SPLiT-TeSGe auxilia a geração de artefatos de teste com uma porcentagem de reutilização considerável para este exemplo. Assim, o nosso processo demonstra um possível ganho quando comparado a abordagens que não consideram reuso como uma estratégia para geração de artefatos de teste onde casos de teste são gerados individualmente para cada produto. Além disso, a utilização de métodos de geração de sequências de teste, tais como o HSI contribui para uma redução no esforço na atividade de teste uma vez que otimiza a geração de casos de teste e permitem a co-

bertura completa das falhas. Estas características são essenciais no contexto SPL, porque o aumento da variabilidade influencia a quantidade de testes necessários para garantir a qualidade do produto. Finalmente, o processo SPLiT-TeSGe pode ser utilizado como um guia para adaptar-se vários métodos, contribuindo para otimizar a atividade teste como um todo.

5. Trabalhos Relacionados

Atualmente, existe um interesse crescente relativo a testes de SPL. Neste contexto, vários estudos apresentam abordagens e metodologias para derivar casos de teste funcionais a partir de modelos de sistemas adaptados para representar informações de variabilidade em SPLs. Alguns exemplos de trabalhos são os apresentados por: McGregor *et al.* [Mcgregor et al. 2002], Bertolino *et al.* [Bertolino and Gnesi 2004], Reuys *et al.* [Reuys et al. 2006] e Capellari *et al.* [CAPELLARI 2012].

Na abordagem proposta por McGregor *et al.* [Mcgregor et al. 2002], casos de teste com informações de variabilidade são derivados na Engenharia de Domínio e reutilizados na Engenharia de Aplicação. No entanto, a abordagem proposta pelo autor não se baseia na técnica MBT, mas centra-se na derivação de casos de teste a partir de documentos de requisitos em linguagem natural. Portanto, essa abordagem não se beneficia das vantagens da adoção de MBT, *e.g.* validação dos requisitos por meio de modelos de teste, geração sistemáica de casos de testes.

Hartmann *et al.* [Hartmann et al. 2004] utiliza diagramas de atividades como modelos de teste e variabilidade. No entanto, os casos de teste são derivados apenas na Engenharia de Aplicação e por isso não se beneficia da reutilização de casos de teste.

Reuys *et al.* [Reuys et al. 2006] propõe um método chamado *Scenario based TEst Case Derivation* (ScenTED) para derivar casos de teste a partir de diagramas de caso de uso atividades. Esses casos de teste são gerados utilizando uma versão estendida do critério *Branch Coverage* no contexto SPL. A representação dos casos de teste é feita através de diagramas de seqüências que contêm dados de teste reais. Embora este método permita a reutilização de casos de teste preservando informações de variabilidade durante a Engenharia de Domínio (bem como o SPLiT-TeSGe), ele não descreve de forma explícita como gerar casos de teste, considerando relações de dependência, tais como, *requires* ou *excludes*. Além disso, para cada seqüência de teste gerada pelo critério *Branch Coverage*, é necessário gerar um diagrama de sequências correspondente, bem como adaptar este diagrama na Engenharia de Aplicação. Por outro lado, SPLiT-TeSGe apresenta uma forma sistemática para adaptar MGSTs no contexto de SPLs. Estes métodos são utilizados para gerar sequências de teste durante a Engenharia de Domínio, as quais são aplicadas para testar produtos na Engenharia de Aplicação.

Outro trabalho similar ao SPLiT-TeSGe é proposto por Capellari [CAPELLARI 2012]. Os autores apresentam o método *FSM-based Testing of Software Product Line FSM-TSPL*, o qual faz uso do método HSI para a geração de sequências de teste. No entanto, diferente do que acontece no SPLiT-TeSGe, este método gera sequências de teste durante a Engenharia de Aplicação e se baseia no reuso oportunístico [Reuys et al. 2006]. Este tipo de reuso não é realizado de forma sistemática, ou seja, os casos de teste são reutilizados de forma *ad-hoc*. Outra desvantagem deste tipo de reuso é que os artefatos podem não ser corretamente gerados para o primeiro produto.

Neste caso, o teste dos demais produtos também será prejudicado. Por outro lado, no SPLIT-TeSGe, as sequências de teste são geradas sistematicamente e não prevê reúso na Engenharia de Domínio.

A maioria destes trabalhos aplicam MBT para gerar casos de teste funcional na Engenharia de Domínio, os quais são reutilizados para testar produtos específicos. Embora estes trabalhos apresentem alternativas que reduzem o esforço de teste, a maioria deles foca apenas no reuso dos artefatos desenvolvidos. Por outro lado, o SPLiT-TeSGe tem por objetivo minimizar este esforço por meio da adaptação de métodos de geração de sequências de teste, *e.g.*, HSI, UIO, W e Wp. Estes métodos permitem maior cobertura dos defeitos e geram sequências de teste menores.

6. Conclusão

Este trabalho apresentou o processo SPLiT-TeSGe que consiste em um processo para adaptação de MGSTs para SPL. Este processo visa gerar sequências com variabilidade na Engenharia de Domínio, a partir de MGSTs, para serem reutilizadas de forma planejada, otimizando o tempo e esforço quando gerados produtos na Engenharia de Aplicação. Além do reuso oportunizado pelo contexto de SPL, a utilização de MGSTs reduz os esforços na realização de testes, pois MGSTs geram sequências de testes que permitam percorrer o menor caminho para efetuar o teste completo de uma aplicação. Com base nos dados apresentados, podemos constatar que a utilização de MGSTs, na realização de teste de Linha de Produto de Software agrega benefícios a quem adota. Foi possível demonstrar que o reaproveitamento das sequências de testes geradas na Engenharia de Domínio, para a reutilização na Engenharia de Aplicação pode atingir índices elevados (87% para o exemplo utilizado). Ainda, deve-se levar em consideração a vantagem de realizar uma única modelagem para testar diversos produtos. Este trabalho está inserido nos esforços para a criação de uma Linha de Produto de Software para geração de ferramentas de testes, chamada PLeTs [Rodrigues et al. 2010]. Agradecemos o apoio financeiro recebido da Dell Computadores do Brasil Ltda, bem como os colegas do Centro de Pesquisa em Engenharia de Software (CEPES), da PUCRS em parceria com a Dell, Flávio Moreira de Oliveira, Élder de Macedo Rodrigues e Maicon Bernardino da Silveira.

Referências

Apfelbaum, L. and Doyle, J. (1997). Model based testing. In *Software Quality Week Conference*, pages 296–300.

Avizienis, A., Laprie, J. C., Randell, B., and Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transaction on Dependable Secure Computing*, 1:11–33.

Bertolino, A. and Gnesi, S. (2004). PLUTO: A Test Methodology for Product Families. In *Software Product-Family Engineering*, pages 181–197.

CAPELLARI, M. L. (2012). Reutilização de teste em linha de produtos de software baseado em máquina de estados finitos para sistemas embarcados. Master's thesis, Universidade Estadual de Maringá, Maringá, Paraná, Brasil.

Chow, T. S. (1978). Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187.

Costa, L. T., Zorzo, A. F., Zanin, A., and Rodrigues, E. M. (2014). Testing in software product lines. Submetido para o Journal of Software: Testing, Verification and Reliability.

Devanbu, P., Karstu, S., Melo, W., and Thomas, W. (1996). Analytical and Empirical Evaluation of Software Reuse Metrics. In *Proceedings of the 18th International Conference on Software Engineering*, pages 189–199.

Fujiwara, S., Khendek, F., Amalou, M., Ghedamsi, A., et al. (1991). Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603.

Hartmann, J., Vieira, M., and Ruder, A. (2004). A UML-based Approach for Validating Product Lines. In *Proceedings of the International Workshop on Software Product Line Testing*, pages 58–65.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie Mellon University.

Kanjilal, S., Chakradhar, S. T., and Agrawal, V. D. (1995). Test function embedding algorithms with application to interconnected finite state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(9):1115–1127.

Luo, G., Petrenko, A., and Bochmann, G. v. (1995). Selecting test sequences for partially-specified nondeterministic finite state machines. In *Protocol Test Systems*, pages 95–110. Springer.

Mcgregor, J. D., Northrop, L. M., Jarrad, S., and Pohl, K. (2002). Initiating Software Product Lines. *IEEE Software*, 19:24–27.

Oliveira, E. A., Gimenes, I. M. S., and Maldonado, J. C. (2010). Systematic Management of Variability in UML-based Software Product Lines. *Journal of Universal Computer Science*, 16:2374–2393.

Reuys, A., Reis, S., Kamsties, E., and Pohl, K. (2006). The ScenTED Method for Testing Software Product Lines. In *Software Product Lines*, pages 479–520.

Rodrigues, E. M., de Oliveira, F. M., Costa, L. T., Bernardino, M., Zorzo, A. F., Souza, S. d. R. S., and Saad, R. (2014). An empirical comparison of model-based and capture and replay approaches for performance testing. *Empirical Software Engineering*, pages 1–30.

Rodrigues, E. M., Viccari, L. D., Zorzo, A. F., and de Souza Gimenes, I. M. (2010). PLeTs-Test Automation using Software Product Lines and Model Based Testing. In *Proceedings of the 22th International Conference on Software Engineering and Knowledge Engineering*, pages 483–488.

Sabnani, K. and Dahbura, A. (1988). A protocol test generation procedure. *Computer Networks and ISDN systems*, 15(4):285–297.

(SEI), S. E. I. (2014). Software Product Lines (SPL). Capturado em: http://www.sei.cmu.edu/productlines/frame_report/benefits.costs.htm.

Weiss, D. M. and Lai, C. T. R. (1999). *Software Product-line Engineering: A Family-based Software Development Process*. Addison-Wesley, Boston, MA, USA.