Uma Abordagem para Projeto, Implementação e Derivação de Testes de Linhas de Produto Web

Heitor Mariano, Uirá Kulesza, Roberta Coelho, Eduardo Aranha

Department of Informatics and Applied Mathematics, DIMAp, Brazil
Federal University of Rio Grande do Norte, UFRN, Brazil

heitormariano@ppgsc.ufrn.br
{uira, roberta, eduardo}@dimap.ufrn.br

RESUMO

Este artigo apresenta uma abordagem para o teste de linhas de produto de software Web. A abordagem oferece: (i) diretrizes para a escolha de estratégias de teste na engenharia de domínio e aplicação; (ii) diretrizes para projeto, implementação e reuso de casos de teste automatizado no nível de unidade, integração e sistema; e (iii) suporte ferramental para gerência, customização e derivação automática de casos de teste usando técnicas de derivação automática de software. A abordagem é avaliada através da sua aplicação em uma linha de produto para sistemas de Comércio Eletrônico (*E-Commerce*).

ABSTRACT

This paper aims to propose an approach for the testing of web software product lines. The approach provides: (i) guidelines for the choice of testing strategies in the domain and application engineering; (ii) guidelines for the design, implementation and reuse of unit, integration and system automated test cases; and (iii) tooling support for management and customization of automated test cases using product line derivation techniques. The approach feasibility is evaluated through its application to an e-commerce web-based product line.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software - Domain Engineering, Reusable Libraries.

General Terms

Design, Languages

Keywords

Web-based Software Product Lines, Software Testing, Automated Testing of Software Product Lines, Web Engineering.

1. INTRODUÇÃO

A adoção de métodos e técnicas de linhas de produto de software (LPS) [3], [12] permite o desenvolvimento de famílias de sistemas relacionados que compartilham características comuns, bem como mantém também suas especificidades. O processo de desenvolvimento de uma LPS se diferencia das abordagens tradicionais, uma vez que apresenta duas etapas essenciais: (i) a engenharia de domínio - quando funcionalidades comuns e variáveis da LPS são definidas e implementadas em torno de uma arquitetura comum de forma a promover o reuso em larga escala; e (ii) a engenharia de aplicação - quando uma ou mais aplicações (produtos específicos) são geradas a partir do reuso dos artefatos criados na engenharia de domínio.

Durante a elaboração da LPS, assim como no desenvolvimento convencional de sistemas, a atividade de teste é fundamental, uma vez que busca melhorar a qualidade do software produzido pela detecção de defeitos nos artefatos de implementação. Além disso, uma abordagem eficaz de teste nos auxilia a reduzir o esforço de desenvolvimento, alcançar os critérios de satisfação do cliente e diminuir os custos de manutenção [7].

No contexto de linhas de produto [3], [12] os testes se manifestam na engenharia de domínio e aplicação. Devido às suas características peculiares, o processo de teste de LPSs apresenta novos desafios que precisam ser considerados, tais como: (i) a escolha adequada dos artefatos que devem ser testados; (ii) a definição e escolha apropriada das estratégias de teste que serão utilizadas; e (iii) a investigação de mecanismos para lidar com as variabilidades nos artefatos de teste. Ao longo dos últimos anos, vários trabalhos relacionados à definição do processo de teste de uma LPS têm sido propostos [1], [8], [11], [12], [16]. Entretanto, tais trabalhos apresentam apenas diretrizes gerais, tais como, motivar o uso de testes para o núcleo da arquitetura da LPS ou apenas dos produtos derivados, assim como promover o reuso de parte dos artefatos de teste criados na engenharia de domínio na engenharia de aplicação. Além disso, esses trabalhos não consideram aspectos peculiares de testes de LPS de domínios particulares, tais como, sistemas web, sistemas embarcados, e aplicações para dispositivos móveis.

O uso de ferramentas que auxiliem o desenvolvimento, automação e acompanhamento de resultados nos testes de uma LPS, como também a gerência e customização dos artefatos de teste por parte dos engenheiros pode ser caracterizado como outra questão de grande importância. Tais ferramentas são fundamentais para permitir a seleção e configuração correta dos artefatos de teste automatizados considerando estratégias específicas. Contudo, ainda é escasso o número de ferramentas que auxiliam em tal atividade, evidenciando a necessidade de mais trabalhos que explorem concretamente meios de ajudar o processo de teste de LPSs.

Nesse contexto, este artigo propõe uma abordagem para testes de linhas de produto Web. Porém, apesar da abordagem poder servir de base para o teste de LPSs implementadas em diversas linguagens, o trabalho é direcionado especificamente para LPSs Web implementadas na plataforma *Java Enterprise Edition* (JEE). A abordagem é formada por um conjunto de atividades que

WebMedia'11: Proceedings of the 17th Brazilian Symposium on Multimedia and the Web. Full Papers.
October 3 -6, 2011, Florianópolis, SC, Brazil.
ISSN 2175-9642.
SBC - Brazilian Computer Society

contemplam: (i) estratégias de testes automatizados; (ii) diretrizes para projeto e implementação de testes na engenharia de domínio e aplicação; e (iii) gerência, customização e derivação automática de variabilidades em artefatos de teste através da extensão de uma ferramenta de derivação de produtos. A viabilidade da abordagem é avaliada neste artigo por meio da sua aplicação em uma linha de produto do domínio de comércio eletrônico (*E-Commerce*).

2. FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta uma visão do processo de teste em LPSs, bem como as características do GenArch, ferramenta estendida para dar suporte a abordagem de teste proposta no artigo.

2.1 Testes em Linhas de Produto de Software

Os testes de LPSs se manifestam tanto na engenharia de domínio como na engenharia de aplicação [12]. Na engenharia de domínio não existem, em geral, aplicações prontas para serem executadas. Dessa forma, os testes são voltados basicamente para componentes individuais e subsistemas da arquitetura formados por componentes comuns a todas as aplicações/produtos. Por outro lado, também é possível analisar como testar componentes da arquitetura que possuem variações. Para alcançar esse objetivo, pode ser usada uma estratégia na qual é construída uma aplicação exemplo e com base nela predefinir artefatos de teste que seriam usados durante os testes da fase de aplicação. Além de testar os artefatos da engenharia de domínio, nessa fase também ocorre o desenvolvimento de testes reusáveis como projetos de teste e dados de teste. Na engenharia de aplicação, por sua vez, são realizados testes adicionais visando encontrar problemas em aplicações e garantir que as variações especificadas para um produto estão presentes, bem integradas ao núcleo da arquitetura e funcionando corretamente. Nessa fase podem também ser empregados testes reusáveis que foram criados na engenharia de domínio, reduzindo os esforços nos testes das aplicações. Contudo, devido à presença de variabilidades e de arranjos específicos dessas variabilidades num produto, a forma como os testes podem ser reaproveitados se caracteriza como uma tarefa não trivial, pois para cada produto gerado, precisa existir a seleção de artefatos de testes específicos para serem aplicados [11]. Por fim, os critérios de cobertura dos testes devem considerar tanto os elementos comuns e variáveis produzidos na engenharia de domínio como as partes específicas de produtos recém criadas.

Um quesito importante considerado no processo de teste de LPS é a gerência de artefatos de teste. Para facilitar os testes é interessante, por exemplo, que as diferenças entre os testes do núcleo e dos produtos sejam conhecidas. A rastreabilidade também é necessária e pode ajudar na seleção adequada de artefatos de teste. Ainda com relação à gerência, repositórios de artefatos de teste são empregados em muitos casos. Eles são usados de maneira que os elementos de teste fiquem separados do código fonte das aplicações ou de forma que os testes estejam integrados [10]. Por fim, existe também a preocupação de usar ferramentas que possam ajudar no processo de teste de uma LPS. Por exemplo, ferramentas que consigam gerenciar testes reusáveis e que facilitem a execução e análise de resultados de teste, como também integrem recursos como repositórios de artefatos de teste com o ambiente de teste adotado na linha. No entanto, a quantidade de ferramentas que suportem adequadamente o teste de linhas de produto é escassa [16].

2.2 Ferramentas de Derivação

No processo de derivação [15] é recomendado a aplicação de ferramentas que promovam a seleção, composição e customização de artefatos definidos durante a engenharia de domínio para a LPS, com o objetivo de derivar produtos específicos. Nesse contexto, se enquadram ferramentas, tais como, o Gears [6] e o pure::variants [13]. Essas ferramentas ajudam na gerência de variabilidades da LPSs e promovem a automatização do processo de geração de produtos com base em configurações e artefatos que implementam as características dos membros da linha. No presente artigo, é adotada uma ferramenta de derivação, chamada GenArch - *Generative Architectures* [2], a qual será apresentada a seguir.

2.2.1 Ferramenta GenArch

O GenArch é uma ferramenta baseada em modelos que permite a derivação de produtos de uma LPS e que foi desenvolvida como um *plug-in* da plataforma Eclipse. Embora outras ferramentas como o Gears e o pure::variants apresentem funcionalidades úteis para auxiliar no desenvolvimento de LPSs e na derivação de seus produtos, elas são bastante complexas para serem empregadas pela comunidade tradicional de desenvolvedores que possuem pouca familiaridade com vários dos novos conceitos da área de LPS. Neste contexto, o GenArch foi desenvolvido com o objetivo de simplificar tanto o processo de representação explícita das variabilidades de LPSs, como para promover a derivação automática de seus produtos.

A ferramenta utiliza três modelos para representar os elementos que compõe uma LPS, são eles: (i) modelo de arquitetura; (ii) modelo de característica; e (iii) modelo de configuração. O modelo de arquitetura permite a representação visual dos artefatos que implementam a LPS, tais como, classes, aspectos, *templates* ou arquivos de configuração. O modelo de característica, como o próprio nome sugere, apresenta as características obrigatórias, alternativas e opcionais da linha de produto. A ferramenta usa como base o modelo de características apresentado em [4]. Por fim, o modelo de configuração define o mapeamento entre as características e os artefatos de implementação.

Para oferecer suporte a derivação de um produto, o GenArch necessita da definição dos seus modelos base, os quais são produzidos através da varredura e identificação de anotações próprias do GenArch escritas na linguagem Java. Elas são usadas para anotar elementos de implementação do projeto Java usado no desenvolvimento da LPS.

Com relação ao uso da ferramenta, a atividade inicial que precisa ser realizada é a anotação dos elementos de implementação. O GenArch suporta dois tipos de anotações: *@Feature* e *@Variability*. A primeira determina a relação entre os artefatos de implementação com as características da LPS. Essa anotação também permite especificar o nome da característica, o tipo (obrigatória, opcional ou alternativa) e seu respectivo pai, caso exista. A segunda é para indicar pontos de variação da arquitetura de uma LPS. Para tratar esses pontos são usados *templates*, os quais são implementados com a linguagem XPand do *plug-in* openArchitectureWare (oAW). Os *templates* são capazes de especificar tanto o código que será usado por todos os membros da LPS como código variável customizado de acordo com informações obtidas do modelo de características ou pelo uso de elementos conhecidos como fragmentos. Esses fragmentos, por sua vez, representam trechos de código ou texto que são

associados a características da LPS e gerenciados pelo modelo de configuração.

Uma vez tendo anotado os artefatos de implementação da LPS, é necessário gerar automaticamente os modelos do GenArch. A ferramenta, através do seu módulo de importação, faz uma varredura nos elementos de implementação e recupera as anotações do tipo *@Feature* para criar a versão inicial dos modelos de arquitetura, característica e configuração, bem como identifica as anotações do tipo *@Variability* para definir os *templates*. Em seguida, os modelos gerados podem ser refinados caso se deseje adicionar algum elemento ou informação extra para a derivação de dado produto. Com todos os modelos finalizados, o GenArch, através do seu módulo de derivação, é capaz de gerar os produtos da LPS. Para criar cada membro da linha, é preciso, inicialmente, definir uma configuração com base no modelo de característica, ou seja, selecionar um conjunto de características que farão parte do produto desejado. Com a configuração do modelo de característica e com as informações dos modelos de arquitetura e configuração, o GenArch é capaz de derivar os produtos. Uma derivação no GenArch resulta na criação de um projeto Java na plataforma Eclipse, que contém todos os artefatos de implementação relacionados as características selecionadas para o produto desejado, tal informação é obtida através do modelo de configuração.

3. UMA ABORDAGEM DE TESTE DE LINHAS DE PRODUTO WEB

Esta seção descreve a abordagem de teste para LPSs web proposta neste artigo. Ela oferece diretrizes claras para os responsáveis pelo processo de teste e para auxiliar a sua aplicação é usada uma extensão da ferramenta GenArch. As subseções seguintes destacam as atividades relacionadas a nossa abordagem que são: (i) escolha das estratégias de teste (Seção 3.1); (ii) estratégias e diretrizes de implementação de testes durante a engenharia de domínio e aplicação (Seção 3.2); e (iii) derivação de artefatos de teste na engenharia de domínio e aplicação (Seção 3.3).

3.1 Escolha de Estratégias de Teste

Uma das atividades da abordagem proposta é a escolha das estratégias que serão usadas para testar a LPS. Para isso, considera-se o documento de requisitos da LPS, os modelos iniciais do GenArch que possam ter sido criados antes do processo de teste e as implementações da linha de produto. Além disso, deve-se levar em conta: (i) decisões de que estratégias gerais serão utilizadas, tais como, teste apenas do núcleo da LPS, teste apenas dos produtos, ou teste de ambos; assim como (ii) a seleção de que tipos e estágios (níveis) de teste podem ser aplicados dado uma estratégia de teste de LPS escolhida. Diversos fatores podem influenciar nessa escolha, tais como: (i) a tecnologia de implementação das variabilidades; (ii) o tipo e domínio do software envolvido na LPS; (iii) o grau de amadurecimento da equipe com automação de testes; e (iv) os custos para sua implementação. Apesar da importância do uso de estratégias para guiar o processo de teste, os trabalhos recentes no contexto de LPS têm proposto estratégias com diretrizes muito gerais [1], [8], [11], [12], [16]. Dessa forma, a nossa abordagem tem o intuito de oferecer diretrizes explícitas para o projeto e implementação de testes de LPSs Web. Para alcançar essa finalidade, são propostas cinco estratégias efetivas de teste, as quais são: (i) testes de unidade do núcleo, (ii) teste de integração do núcleo, (iii) teste de sistema do núcleo, (iv) teste de unidade e integração de produtos e (v) teste de sistema de produtos.

3.2 Estratégias e Diretrizes de Implementação de Testes

A abordagem proposta contempla cinco estratégias de teste conforme destacado na Seção 3.1. As subseções seguintes detalham cada uma dessas estratégias da abordagem.

3.2.1 Teste de Unidade do Núcleo

Durante a engenharia de domínio, a abordagem motiva a criação de testes de unidade para as classes relevantes ou críticas do núcleo da LPS. Caso problemas existam nessas classes, eles podem ser corrigidos, impedindo que sejam transferidos para os produtos derivados da LPS. Na implementação das classes de teste, é usado o framework de teste de unidade e integração JUnit, usado comumente pela comunidade. O JUnit pode ser aplicado tanto para o desenvolvimento e execução dos testes como para a visualização dos resultados obtidos de maneira simples e intuitiva. A abordagem propõe também o uso de *Mock Objects* [5]. Os *Mock Objects* ou simplesmente *mocks* são implementados com auxílio do *framework* jMock [5].

A abordagem lida com dois cenários de teste de unidade: (i) testes de unidade sem *mocks*; e (ii) testes de unidade com *mocks*. No primeiro caso, os testes usam apenas classes reais do núcleo da LPS. No segundo, os *mocks* são usados para substituir um grupo de classes do núcleo que interagem com a classe do núcleo que será testada individualmente. Os *Mock Objects*, em particular, são usados para simular objetos reais que apresentam um comportamento complexo ou ainda bastante impreciso, difíceis de serem inicializados por dependerem de informações de uma base de dados ou por simplesmente ainda não existirem. Além disso, com o uso de *mocks* é possível definir expectativas na execução de métodos e meios para verificá-las.

3.2.2 Teste de Integração do Núcleo

Na engenharia de domínio, a abordagem também apresenta diretrizes para a criação de testes de integração para avaliar o funcionamento e colaboração entre objetos do núcleo da LPS. Tais testes são implementados com o apoio do JUnit e podem pertencer aos seguintes cenários: (i) testes de integração sem variabilidades; e (ii) testes de integração com a presença de pontos de variação. Na primeira situação, existe o desenvolvimento de testes para classes do núcleo que interagem com outras classes do núcleo. No segundo caso, os testes são criados para classes do núcleo que colaboram tanto com outras classes do próprio núcleo como com classes que implementam variações. Nesse cenário específico, são usados *mocks* para simular uma parte das classes que implementam variabilidades da linha. Dessa forma, o código de classes referente ao núcleo pode ser testado sem que seja preciso usar todos os objetos concretos das variações que elas dependem. Para a definição dos *mocks* nos testes de integração, o framework jMock é novamente usado.

3.2.3 Teste de Sistema do Núcleo

Testes de sistema para o núcleo também são encontrados na abordagem. Como na engenharia de domínio, em geral, não existem aplicações Web prontas para serem executadas, nossa proposta é que nessa etapa os testes do núcleo sejam parcialmente implementados na forma de *templates* de geração de código de teste, os quais serão usados na engenharia de aplicação. No caso específico da nossa abordagem, os *templates* são usados para criar *scripts* em formato HTML para o *Selenium IDE* [14], ferramenta para testes de sistema web. Com relação à forma de implementação dos *templates*, existem dois cenários possíveis: (i)

templates para páginas Web obrigatórias e de conteúdo fixo; (ii) *templates* para páginas Web de conteúdo fixo e variável. No primeiro caso, os *templates* não lidam com as variabilidades da linha e são formados apenas por código de teste comum a todas as páginas das aplicações (produtos) Web. No segundo caso, cada *template* contempla: (i) uma parte fixa que representa o código de *script* de teste que se mantém o mesmo independente das variabilidades escolhidas para dado produto; e (ii) uma parte variável que contempla partes de código do *script* a serem geradas de acordo com as variabilidades definidas para um produto.

3.2.4 Teste de Unidade e Integração de Produtos

Na engenharia de aplicação, os testes de unidade e integração para o núcleo podem ser re-executados como testes de regressão. Além de poder reusar os testes de unidade do núcleo, novos testes desse estágio podem ser criados, mas voltados para classes que representam variantes concretas dos pontos de variação. No que se refere aos testes de integração, é preciso adaptar os testes de integração do núcleo para que sejam aplicados entre os membros gerados na LPS. As adaptações envolvem a retirada dos *mocks* usados nos testes do núcleo para dar lugar às implementações reais das variações que as classes do núcleo dependem. Para alcançar esse objetivo, são usados *templates* que são customizados pelo GenArch, de forma a controlar a inclusão ou não de classes reais devido as variações que essas classes implementam fazerem parte ou não de um dado produto. Além disso, os *templates* são usados para alcançar reuso. O código comum de tais arquivos está sempre presente entre os testes de integração dos produtos, enquanto a parte variável é passível de reutilização, uma vez que pode existir um subconjunto entre todos os membros da LPS que compartilham uma mesma variabilidade. Dessa forma, o trecho de código para testá-la estará no arquivo de teste de cada produto do subconjunto.

3.2.5 Teste de Sistema de Produtos

A abordagem também delimita testes do nível de sistema para os produtos. Conforme visto na Seção 3.2.3, durante a engenharia de domínio, *templates* são implementados para dois casos específicos: (i) testar páginas Web obrigatórias que tem apenas conteúdo comum; e (ii) testar páginas Web que possuem conteúdo tanto comum como variável. Na engenharia de aplicação, os *templates* são processados e originam os *scripts* para o *Selenium IDE*. Os *scripts* gerados pelos *templates* do primeiro caso são sempre reaproveitados, pois foram projetados para testar páginas que serão carregadas para todos os produtos da LPS. Com relação aos *scripts* cridos pelos *templates* do segundo caso, eles são formados por uma parte comum que é reutilizada e uma parte variável que também pode ser reusada, desde que a variabilidade que condiciona a entrada de tal parte variável esteja em mais de um produto.

3.3 Derivação Automática de Testes

A derivação dos testes dos produtos pode ser iniciada quando as classes de teste estão devidamente anotadas, bem como os modelos da extensão do GenArch prontos. Com relação a anotação dos testes, é usado na abordagem a anotação *@Feature* para relacionar os testes com características da LPS. Além dessa anotação, foi incluído na nossa nova extensão do GenArch, uma nova anotação *@Test* para indicar: (i) a estratégia de teste, isto é, definir se os testes são para o núcleo ou para os produtos da linha; (ii) o estágio (unidade, integração, sistema) que os testes pertencem; e (iii) o tipo que, no momento, pode ser apenas funcional.

Com os artefatos de teste anotados com *@Test*, a extensão do GenArch, no momento da geração dos modelos, identifica tais artefatos e os inclui no modelo de arquitetura. As informações referentes a estratégia, o estágio e o tipo que o teste anotado pertence também estarão visíveis no modelo. Existindo também a presença de *@Feature* numa classe de teste, tal classe estará representada no modelo de configuração. As informações definidas em *@Test* estarão acessíveis, bem como a indicação da característica que o teste está associado. O modelo de características, por sua vez, não sofreu alteração na extensão da ferramenta GenArch.

Com os modelos finalizados, a derivação automática pode ser realizada. A extensão do GenArch implementa duas formas de derivação: (i) geração de código de teste para o núcleo; e (ii) geração de código de teste para produtos. Na primeira forma, o usuário define por meio de um assistente, o nome do projeto para a derivação e o estágio (unidade, integração e sistema) dos testes que precisam ser carregados. No final do processo, são derivados exclusivamente os elementos de implementação obrigatórios com os testes voltados para esses artefatos do núcleo e pertencentes aos estágios definidos. Na segunda forma, além da definição do projeto de derivação e dos estágios, o usuário seleciona a configuração de um produto. Dessa forma, é possível saber as características comuns e variáveis que o constituem, e com as associações definidas em tais características com os elementos de implementação no modelo de configuração, o algoritmo de derivação pode selecionar os artefatos específicos para o membro da LPS. Os testes, por sua vez, são carregados de acordo com a configuração do produto e dos estágios de teste selecionados, ou seja, se determinada característica estiver presente, os artefatos de implementação e teste relacionados a ela são derivados.

4. APLICAÇÃO DA ABORDAGEM

Esta seção apresenta uma LPS Web do domínio de comércio eletrônico (*E-Commerce*), bem como destaca a aplicação da abordagem de teste proposta por meio da exposição de quatro atividades: (i) escolha de estratégias de teste; (ii) projeto e implementação de testes; (iii) preparação de artefatos de teste para derivação; e (iv) derivação automática de testes.

4.1 Linha de Produto *E-Commerce*

Com a finalidade de avaliar a nossa abordagem é usado como estudo de caso uma LPS Web desenvolvida em Java e pertencente ao domínio de comércio eletrônico. Grande parte da análise de domínio dessa linha de produto foi fornecida no trabalho feito por [9]. As características da linha são divididas em dois grupos principais: *Store Front* e *Business Management*. *Store Front* reúne características referentes às interfaces de interação dos clientes com as aplicações (produtos) da LPS *E-Commerce*. *Business Management* apresenta características relacionadas com as operações de negócio como gerenciamento de produtos e pedidos dos clientes, *marketing*, entre outras operações que colaboram no funcionamento de uma loja virtual.

A LPS *E-Commerce* foi desenvolvida usando uma arquitetura MVC (*Model - View - Controller*). Na camada *Model* da LPS estão presentes os *Session Beans* que possuem a implementação de negócio, as classes de domínio da arquitetura da LPS e as classes de acesso a dados (DAOs). Na camada *View* existem as páginas web implementadas na tecnologia *Java Server Pages* (JSP) que com o auxílio do *framework Java Server Faces* (JSF) renderizam os dados de visualização. Por fim, na camada

Controller estão os *Managed Beans*, entidades que funcionam como controladores e interfaces de comunicação entre as camadas *View* e *Model*.

4.2 Escolha de Estratégias de Teste

Na LPS *E-Commerce* foram aplicadas as diretrizes para implementação de estratégias de teste tanto para os artefatos do núcleo quanto para os produtos. Para tornar a atividade de teste a mais completa possível, foram usadas na LPS todas as cinco estratégias concretas de teste da abordagem deste artigo (ver Seção 3.2). Em especial, é ilustrado como testes para o núcleo podem ser automaticamente reusados para testes de produtos específicos. Os artefatos de teste foram desenvolvidos em duas fases. Na primeira, os casos de teste foram elaborados para os artefatos produzidos na engenharia de domínio da LPS. Na segunda, os testes foram voltados para os produtos gerados na engenharia de aplicação. Por fim, as atividades de teste também buscaram verificar problemas nos artefatos comuns ou mesmo variáveis já testados anteriormente nos produtos recém criados e que ainda não tinham passado por testes.

4.3 Projeto e Implementação de Testes

Na LPS *E-Commerce* foram implementados testes de unidade e integração para o núcleo. Analisando primeiramente os de unidade, foram criados dois testes que não utilizam *mocks*. Um deles é para a classe `GenericDAOImpl` que faz parte da camada de persistência e que é herdada pelas classes de acesso a dados (DAOs). O outro é para a classe `ProdutoDAO` que realiza operações de cadastro e de manipulação dos dados dos produtos das aplicações da LPS *E-Commerce*. No que se refere aos testes de unidade com *Mock Objects,* foram desenvolvidas seis classes de teste para classes de serviço que auxiliam, entre outras funções, no cadastro e exclusão de produtos, bem como na aprovação de tais produtos para venda.

Com relação aos testes de integração, foi criado um caso de teste para a classe `ProdutoMBean` que é da camada de controle da LPS e que participa do cadastro e busca de produtos, assim como na definição de informações extras e liberação desses produtos para comercialização.

Os testes de sistema foram definidos durante a engenharia de domínio para páginas responsáveis pelas seguintes funções: (i) cadastro de tipo de produtos; e (ii) cadastro geral de produtos. No primeiro caso, foi implementado, por exemplo, o *template* `TesteCadastroTipoProduto.xpt`. A Figura 1 apresenta uma parte de tal arquivo.

```
1 ...
2 <tr>
3     <td>open</td>
4     «LET projectName() AS name»
5     <td>/«name»/dynamicContent.jsf</td>
6     «ENDLET»
7     <td></td>
8 </tr>
9 ...
10 <tr>
11     <td>clickAndWait</td>
12     <td>link=Cadastro Categoria de Produto</td>
13     <td></td>
14 </tr>
15 ...
```

Figura 1. Parte do *template* TesteCadastroTipoProduto.xpt.

O *template* da Figura 1 não lida com as variabilidades de um dado produto, uma vez que a página para registro dos tipos de produtos possui apenas código fixo. Na figura, é ilustrado também o uso do método `projectName()` (linha 4), criado na extensão do GenArch. Esse método recupera o nome do projeto destinado a conter o código e os testes de um produto derivado, que por sua vez é usado para customizar o nome do projeto web que será chamado pelo script do Selenium. O restante do conteúdo do *template* representa a parte comum do *script* de teste. Para o teste de cadastro de produto, por sua vez, foi criado o *template* `TesteCadastroProduto.xpt`. Tal *template* é capaz de controlar a entrada ou não de trechos variáveis do *script* de teste dependendo das variabilidades de dado produto. A Figura 2 apresenta uma parte do arquivo `TesteCadastroProduto.xpt`.

```
1 ...
2 «LET featureConfigurations("ProductInformation",featureInstance)
3  AS iterator»
4  «FOREACH iterator.features AS child»
5  «IF child.isSelected»
6      «IF child.name == "WarrantyInformation"»
7 <tr>
8     <td>type</td>
9     <td>cadastroproduto:warranty</td>
10    <td>1</td>
11 </tr>
12 ...
```

Figura 2. Parte do *template* TesteCadastroProduto.xpt.

No *template* é também usado o método `productName()` para inserir o nome do projeto de um produto gerado na LPS, mas na Figura 2 é dada atenção ao método `featureConfigurations()`, que permite obter informações sobre características selecionadas no modelo de característica do produto sendo gerado. No *template*, esse método é usado para recuperar as características ligadas a `ProductInformation` (linha 2). No grupo de características obtidas existem as seguintes características opcionais (variabilidades): *DetailedDescription*, *Size*, *Weight*, *WarrantyInformation* e *Availability*. A presença dessas características implica em mudanças no formulário de cadastro. Dessa forma, é fundamental o *template* conseguir customizar adequadamente o *script* de teste com base na existência ou não de tais variabilidades. Assim o código do *template* analisa se certa característica foi selecionada (linha 5) e caso tenha sido selecionada é verificado se o nome da característica é igual a *WarrantyInformation* (linha 6). Caso a condição seja verdadeira, é inserido no *script* final de uma aplicação derivada, o trecho para testar a definição da informação de garantia de um produto que será cadastrado (linhas 7 a 11). A mesma verificação é realizada para *Size*, *Weight* e *DetailedDescription*. Com a presença dessas variações num produto, são incluídos trechos de teste específicos relacionados a cada uma delas.

Os testes de unidade dos produtos são formados pelos testes de unidade do núcleo que são re-executados na engenharia de aplicação. Para os testes de integração dos produtos, foi definido um *template* com base no teste de integração para a classe do núcleo `ProdutoMBean` que participa da função de cadastro de produtos para venda. O *template* em questão adapta o código de teste devido a presença ou não da característica opcional *ContentApproval*. Com a seleção dessa característica é inserido, entre outras definições, a classe `ProdutoSBeanComAutorizacao`, envolvida nas autorizações dos produtos para estarem disponíveis para compra, bem como classe sempre encontrada na implementação dos produtos quando *ContentApproval* está presente. Caso essa variabilidade não faça parte de um produto, é definido nos testes, por exemplo, um *Mock Object* para simular uma instância real de `ProdutoSBeanComAutorizacao`.

Os testes de sistema dos produtos são formados pelos *scripts* para o *Selenium IDE* gerados a partir dos *templates*. A Figura 3 ilustra um *script* HTML criado pelo processamento do *template* `TesteCadastroProduto.xpt`. O *script* foi gerado com base num produto derivado que possui na sua configuração as variações *WarrantyInformation* e *Size*, características que permitem que sejam definidas informações de garantia e tamanho de um produto durante o seu cadastro. Devido à existência de tais variabilidades, são incluídos trechos de código específicos no *script* de teste de sistema (destacado na figura).

```
1<?xml version="1.0" encoding="UTF-8"?>
2...
3<tr>
4    <td>type</td>
5    <td>cadastroproduto:preco</td>
6    <td>10.0</td>
7</tr>
8<tr>
9    <td>type</td>
10   <td>cadastroproduto:warranty</td>
11   <td>1</td>
12</tr>
13<tr>
14   <td>type</td>
15   <td>cadastroproduto:size</td>
16   <td>10</td>
17</tr>
18...
```

Figura 3. *Script* de teste de sistema para um produto.

4.4 Preparação de Artefatos de Teste para Derivação

A preparação dos testes para derivação é iniciada com a anotação das classes de teste da LPS. A Figura 4 apresenta um exemplo de uso da nova anotação *@Test* da extensão do GenArch.

```
...
@Test(strategy=TestStrategy.core,stage=TestStage.unit,
      type=TestType.functional)
public class GenericDAOImplTest extends TestCase {
...
}
```

Figura 4. Classe de teste anotada.

O teste da Figura 4 é voltado para a classe de persistência `GenericDAOImpl` que faz parte do núcleo da LPS *E-Commerce*. A anotação *@Test* usada indica que a classe de teste pertence à estratégia *core*, ao estágio *unit* e ao tipo *functional*. Ela também é empregada para as outras classes de teste. Os valores dos atributos, por sua vez, são definidos com base na estratégia, estágio e tipo que cada teste está enquadrado. Na abordagem é proposto também o uso da anotação *@Feature* para relacionar os testes com as características da LPS, permitindo que ao selecionar certa característica para um produto, o teste para tal característica seja também carregado na derivação. Contudo, não foi necessário o uso de *@Feature* na LPS *E-Commerce* devido a maioria das classes de teste criadas serem do núcleo. Dessa forma, para carregar tais testes é preciso apenas o uso de *@Test*. Além disso, os testes que envolvem variações (teste de integração e sistema de produtos) são implementados como *templates* de teste, não existindo também, nesse caso, a necessidade de empregar a anotação *@Feature*.

Com as classes que implementam a LPS e os testes anotados, o passo seguinte para a preparação dos artefatos de teste é gerar os modelos da extensão do GenArch. Para acessar esse recurso é preciso clicar com o botão direito no projeto, escolher a opção "Genarch" e, em seguida, "Create Models". A partir desse momento, a ferramenta cria um projeto Eclipse com um diretório *models* contendo os modelos de arquitetura, configuração e características. A Figura 5 ilustra parte do modelo de arquitetura da LPS web.

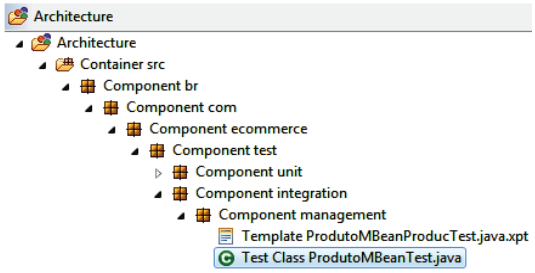

Figura 5. Modelo de Arquitetura.

Na Figura 5 pode ser observado, por exemplo, o *template* `ProdutoMBeanProducTest.java.xpt`, mencionado na Seção 4.3 que define o teste de integração da classe `ProdutoMBean` (elemento do núcleo) e que determina, entre outras customizações, a entrada da classe `ProdutoSBeanComAutorizacao` quando a variabilidade *ContentApproval* é selecionada para um produto. Além disso, é visto o teste de integração `ProdutoMBeanTest` que também testa `ProdutoMBean`, sem a presença de classes que implementam variações, ou seja, é um teste que envolve apenas classes do núcleo. `ProdutoMBeanTest` também serviu de base para a construção do *template* apresentado no modelo.

Como apresentado neste artigo, o modelo de configuração da extensão do GenArch pode representar a associação de testes com características da LPS. Contudo, como tal modelo é usado, essencialmente, para indicar a relação de elementos de implementação com características variáveis e pelo fato de não terem sido criados testes voltados diretamente para características desse tipo, não foi necessário no contexto de teste da linha *E-Commerce* a geração de modelos de configuração contendo elementos de teste.

4.5 Derivação Automática de Testes

A extensão do GenArch desenvolvida oferece suporte para duas formas de derivação: (i) carga dos testes do núcleo durante a engenharia de domínio; (ii) carga dos testes dos produtos durante a engenharia de aplicação. Para iniciar a primeira forma, é preciso clicar com o botão direito no projeto da LPS, escolher a opção "Genarch" e depois "Derivate Core Tests". A Figura 6 apresenta o assistente que é exibido após a escolha dessa opção.

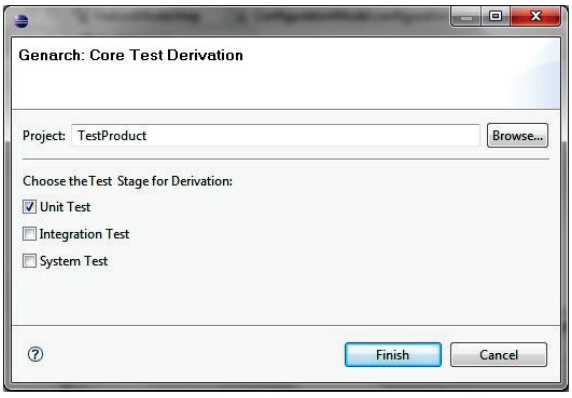

Figura 6. Assistente de derivação dos testes do núcleo.

No assistente é definido o projeto para a derivação e o(s) estágio(s) de teste desejado(s). No final do processo, são carregados os artefatos de código que representam o núcleo da linha, assim como os testes automatizados para os artefatos de implementação do núcleo que pertencem aos estágios de teste escolhidos. Por exemplo, se for especificado a derivação de testes de unidade para o núcleo são carregados os testes anotados com *@Test*, com o atributo *strategy* igual a "core*"* e o atributo *stage* com valor "unit*"*. A Figura 7 ilustra alguns testes unitários para o núcleo carregados no projeto Java de um produto da LPS web.

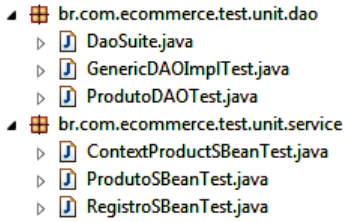

Figura 7. Testes de unidade do núcleo.

Para iniciar a segunda forma de derivação, é necessário clicar com o botão direito no projeto da linha, selecionar a opção "Genarch" e, em seguida, escolher "Derivate Product Tests". A Figura 8 ilustra o assistente que é apresentado depois da definição dessa opção.

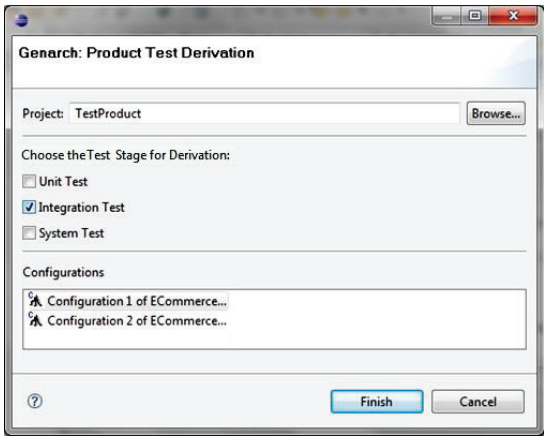

Figura 8. Assistente de derivação dos testes de um produto

O assistente da Figura 8 é semelhante ao da derivação de testes para o núcleo. A diferença está apenas na definição da configuração de um produto. Com a conclusão da derivação, são carregados os artefatos de código relacionados às características da configuração do produto, assim como as classes de teste do núcleo e os testes que lidam com variabilidades da LPS. Os testes que devem ser considerados são os que pertencem aos estágios selecionados pelo usuário da extensão do GenArch. A Figura 9 apresenta testes que são carregados ao definir o estágio de integração para a derivação (opção *integration test*).

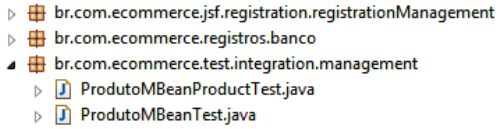

Figura 9. Testes de integração em projeto de Produto.

A Figura 9 apresenta, por exemplo, a classe de teste de integração `ProdutoMBeanTest` para testar `ProdutoMBean`, classe do núcleo da LPS. Existe também a presença da classe de teste `ProdutoMBeanProductTest` que foi criado a partir do *template* `ProdutoMBeanProductTest.java.xpt`. O conteúdo final de tal teste é definido com base na presença ou ausência da característica variável *ContentApproval*.

Na LPS *E-Commerce*, a viabilidade de uso da abordagem foi comprovada. As diretrizes da abordagem ajudaram a direcionar o desenvolvimento dos testes na engenharia de domínio e aplicação, bem como pode definir testes de diversos níveis (unidade, integração e sistema) para os artefatos da LPS. A abordagem demonstrou suprir a necessidade de testar apenas o núcleo da LPS, assim como prover mecanismos para testar tanto as partes comuns como variáveis da linha Web. A extensão do GenArch foi capaz de gerenciar artefatos de teste pela utilização da nova anotação *@Test* e representações específicas no modelo de arquitetura. A extensão oferece também a representação de testes no modelo de configuração, mas para o contexto de teste da LPS *E-Commerce*, esse recurso não foi utilizado. Por fim, os produtos puderam ser derivados com seus testes que, por sua vez, foram carregados com base na estratégia (núcleo ou produto) e estágio de teste (unidade, integração e sistema) definidos pelo usuário da extensão do GenArch.

5. TRABALHOS RELACIONADOS

Diversos trabalhos de pesquisa no contexto de teste de LPS têm sido propostos nos últimos anos. O trabalho elaborado por [8] apresenta uma abordagem de teste de linhas de produto dirigida a riscos que visa oferecer suporte ao planejamento e preparação de testes de LPSs, bem como fornecer técnicas específicas para desenvolver e gerenciar casos de teste para componentes genéricos que os membros da linha de produto dependem. Contudo, nesse trabalho não existem orientações de como usar a abordagem num contexto prático e em domínios específicos (ex: sistemas Web), bem como a gerência dos artefatos dos testes não é explorada em profundidade. Existe carência também na especificação de como conduzir a criação e uso de artefatos de teste. Na abordagem proposta neste artigo, encontram-se diretrizes claras para implementar e gerenciar testes de LPS Web com base em estratégias de teste específicas.

O trabalho de [16] apresenta quatro estratégias gerais de teste que podem ser adotadas para LPSs de diferentes domínios, uma vez que independem do contexto específico de cada LPS. As estratégias são: (i) *teste de produtos* que busca testar os produtos de forma individual, ou seja, a cada novo membro gerado, novos testes são desenvolvidos; (ii) *testes incrementais de linhas de produto* que tem como objetivo testar o primeiro produto gerado de forma individual e à medida que novos produtos são criados, testes de regressão são aplicados; (iii) *instanciação de artefatos reusáveis* que defende a criação de testes o quanto antes possível na engenharia de domínio e na engenharia de aplicação reusar e refinar tais artefatos para atender as necessidades de teste de um produto específico (iv) *divisão de responsabilidades* que separada o processo de teste de uma LPS entre a engenharia de domínio e aplicação. Na nossa abordagem de teste de LPS foram adotadas as seguintes estratégias: (i) reuso de artefatos, visando permitir que testes como os de unidade e integração criados na engenharia de domínio pudessem ser reusados na engenharia de aplicação; (ii) testes incrementais, uma vez que foi preciso assegurar através de técnicas de regressão se o que funciona num produto continua funcionando quando presente em outros produtos da LPS; e (iii) divisão de responsabilidades, pois os testes se manifestam nas duas fases de uma linha de produto. Em outras palavras, nosso trabalho representa uma abordagem concreta que atende as

diretrizes gerais apresentadas em [16], oferecendo assim mais sistematização para o teste de LPSs pertencentes ao domínio Web.

Com relação ao trabalho realizado por [1] é apresentada a metodologia PLUTO (do inglês *Product Line Use Case Test Optimization*) que objetiva derivar cenários de uso dos produtos da LPS e que são testados para verificar se os requisitos foram atendidos. A derivação de tais cenários é feita a partir de requisitos expressos como PLUCs (do inglês *Product Line Use Case*), os quais são construídos com base em linguagem natural estruturada. Apesar dos cenários serem úteis para analisar a adequação dos requisitos dos membros da LPS, eles não se configuram como artefatos executáveis, assim demandando a criação de casos de teste reais para os produtos. Além disso, em [1] não se encontram diretrizes claras de como conduzir o processo de teste de uma LPS com o PLUTO. Na abordagem deste artigo, diferentemente do PLUTO, contempla o aspecto de implementação e uso de testes reais, bem como delimita o momento que tais artefatos são aplicados, como implementá-los e os objetivos que precisam ser satisfeitos. A nossa abordagem fornece também informações detalhadas para orientar os envolvidos em testes de LPSs, em especial, para as linhas de produto Web.

Com base nas análises realizadas, existe uma deficiência no que se refere a diretrizes claras de teste que sirvam de base para auxiliar a replicar os trabalhos desenvolvidos em situações reais de teste de LPSs. Além disso, existe uma forte carência no que se refere ao suporte ferramental para a derivação de estratégias de teste de LPS. A abordagem proposta deste artigo, por sua vez, apresentou extensões implementadas para a ferramenta GenArch que oferecem suporte para a gerência, customização e derivação de artefatos de teste.

6. CONCLUSÃO E TRABALHOS FUTUROS

Este artigo apresentou uma abordagem que objetiva ajudar os envolvidos no processo de teste de LPSs Web implementadas na plataforma *Java Enterprise Edition* (JEE) pela definição de cinco estratégias concretas de teste que são: (i) testes de unidade do núcleo, (ii) teste de integração do núcleo, (iii) teste de sistema do núcleo, (iv) teste de unidade e integração de produtos e (v) teste de sistema de produtos. Para cada estratégia foram fornecidas diretrizes que demonstram uma visão de quais elementos de uma LPS Web podem ser testados e como tais testes podem ser aplicados na engenharia de domínio e aplicação. Além disso, foram desenvolvidas extensões para uma ferramenta de derivação existente, permitindo: (i) definição de informações relevantes (estratégia, estágio e tipo) referentes aos testes pelo uso da nova anotação *@Test*; (ii) representação de elementos de teste nos modelos de arquitetura e configuração; e (iii) carregamento de testes (classes ou *scripts* do *Selenium IDE*) específicos para o núcleo ou para produtos de dada LPS Web. Por fim, o trabalho também pode buscar a viabilidade de uso da abordagem de teste proposta através da sua aplicação numa LPS web para sistemas *E-Commerce*.

No que se refere a trabalhos futuros, pretende-se realizar extensões na abordagem explorada no artigo para o contexto de testes não-funcionais de sistemas web, tais como, teste de desempenho, *stress* e segurança, bem como conduzir experimentos e estudos de caso comparativos da abordagem com outras estratégias de teste propostas. Por fim, outro trabalho futuro que desejamos conduzir é o desenvolvimento de novas extensões no GenArch para que a ferramenta seja capaz de auxiliar o processo de teste do núcleo do *middleware* Ginga de TV Digital que apresenta o seu código escrito em C++. Por tal *middleware* se tratar de uma LPS com características diferentes da linha de produto *E-Commerce*, novas estratégias de teste poderão também ser elaboradas. Assim como o uso da LPS de TV Digital como estudo de caso, irá permitir avaliar a nossa abordagem em um diferente domínio.

Agradecimentos

Este trabalho é parcialmente financiado pelo projeto 134173/2010-4 do CNPq, e pelo projeto PROCAD 090/2007 da CAPES.

7. REFERÊNCIAS

[1] Bertolino, A., and Gnesi, S. Use case-based testing of product lines. ACM SIGSOFT Notes, 2003.

[2] Cirilo, E., Kulesza, U., Lucena, C. A Product Derivation Tool Based on Model-Driven Techniques and Annotations. Jounal of Universal Computer Science 14(8): 1344-1367 (2008).

[3] Clements, P., and Northrop, L. Software Product Lines: Practices and Patterns, Addison-Wesley Professional, 2001.

[4] Czarnecki, K., Eisenecker, U.: "Generative Programming: Methods, Tools, and Applications", Addison-Wesley, 2000.

[5] Freeman, S., Mackinnon, T., Pryce, N., and Walnes, J. jMock: Supporting Responsibility-Based Design With Mock Objects. OOPSLA'04, Oct. 24-28, 2004, Vancouver, British Columbia, Canada.

[6] Gears/BigLever. URL: http://www.biglever.com. Acessado em 21/06/2010.

[7] Juristo, N., and Moreno, A. M. Guest editors' introduction: Software testing practices in industry. IEEE Software, 2006.

[8] Kolb, R. A risk-driven approach for efficiently testing software product lines. GPCE - 5th Generative Programming and Component Engineering, 2003.

[9] Lau, S. Q. "Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates". MSc Dissertation. Waterloo, Ontario, Canada, 2006.

[10] McGregor, J. D. A Practical Guide to Testing Object-Oriented Software. Addison Wesley, 2001.

[11] Pohl, K., and Metzger, A. Software product line testing. Communications of the ACM, 2006.

[12] Pohl, K., Böckle, G., et al. Software Product Line Engineering: Foundations, Principles and Techniques, Springer, 2005.

[13] Pure::Variants. URL: http://www.pure-systems.com. Acessado em 21/06/2010.

[14] Selenium IDE. URL: http://seleniumhq.org/docs/. Acessado em 12/06/2010.

[15] Sybren, D., Marco, S., et al. "Product derivation in software product families: a case study." J. Syst. Softw. 74(2): 173-194, 2005.

[16] Tevanlinna, A., Taina, J., and Kauppinen, R. Product family testing: a survey. ACM SIGSOFT Software Engineering Notes, 29(2):12, 2004.