Automação de Métodos e Técnicas para Teste Funcional de Componentes[1]

Mestrando: *Daniel Lima Barbosa*[2]
Orientadora: *Patrícia Duarte de Lima Machado*
Co-Orientador: *Jorge César Abrantes de Figueiredo*
{daniel, patricia, abrantes}@dsc.ufcg.edu.br

Coordenação de Pós-Graduação em Informática – COPIN
Departamento de Sistemas e Computação – Universidade Federal de Campina Grande
Av. Aprígio Veloso, 882, Caixa Postal: 10.106, CEP: 58.109-970, Campina Grande – PB
Ano de Ingresso: *2003* – Previsão de Conclusão: *Novembro de 2004*
Data da aprovação da proposta de dissertação: *Agosto de 2003*

Resumo

O desenvolvimento de software com padrão de qualidade é hoje um dos maiores desafios da indústria. Em função disto, o interesse no software baseado em componentes tem crescido de forma substancial devido à promessa de qualidade e redução de custos e tempo de desenvolvimento. Entretanto, o reuso efetivo de componentes está fortemente relacionado à confiabilidade dos mesmos. Teste é uma das formas de verificação que mais vem sendo utilizadas na prática. Particularmente, o teste funcional, por se basear na especificação do software, desponta como uma técnica capaz de amenizar os custos inerentes ao processo de teste uma vez que casos de teste podem ser obtidos paralelamente ao seu desenvolvimento. Em função disto, muitas pesquisas estão sendo desenvolvidas com o objetivo de produzir técnicas efetivas para a derivação de casos de teste a partir da especificação dos sistemas. Um ponto de convergência entre os pesquisadores e especialistas da área é a necessidade de automação destas técnicas de forma que todo o processo de teste possa ser executado (e re-executado) com a menor interferência humana possível. O principal objetivo deste trabalho é a automação de dois métodos de teste que, a partir da especificação UML (Unified Modeling Language) do sistema, permitem a geração, execução e análise de resultados de casos de teste funcional e de integração funcional de componentes de software. Para tanto será feito um aperfeiçoamento destes métodos no que se refere a aspectos de automação assim como o desenvolvimento da ferramenta de suporte SPACES.
Palavras-chave: Teste Funcional, Automação de Teste, Componentes, UML.

[1]Este trabalho é financiado pelo CNPq - Projeto MOBILE, processo 552190/2002-0.
[2]O autor recebe apoio financeiro da CAPES.

1 Caracterização do Problema

Na busca pela qualidade do software, processos de teste bem planejados e executados são cada vez mais necessários para garantir que o produto final do desenvolvimento seja o mais livre de falhas possível. Para tanto é preciso que as atividades de teste sejam realizadas cada vez mais cedo, de forma a minimizar o impacto e a propagação dos erros. Neste contexto, o teste funcional surge como uma técnica interessante, uma vez que, a partir da especificação do software, casos de teste podem ser obtidos paralelamente ao seu desenvolvimento.

Aliada a esta busca por processos de teste mais eficientes, a composição de sistemas a partir de componentes também desponta como um fator de determinação de software de alta qualidade, rápida construção e baixo custo. Para tanto os componentes precisam ser testados de forma a aferir a sua confiabilidade e, conseqüentemente, a confiabilidade do sistema como um todo. Do ponto de vista de um componente, os testes a serem realizados precisam satisfazer duas expectativas distintas. Do lado do fornecedor, é preciso garantir que o componente se comportará adequadamente sob os mais diferentes contextos em que venha a ser utilizado. Do lado do cliente, é necessário assegurar que o componente apresente a funcionalidade desejada quando integrado aos demais componentes para compor uma aplicação.

Em função disto, métodos para a verificação funcional de componentes e da integração destes na composição de sistemas vem sendo propostos [4, 7, 6, 3]. Em particular, em [4] é proposto um método de teste funcional aplicável a componentes de software. A principal contribuição desse trabalho está em possibilitar a verificação de propriedades individuais dos componentes e empacotar artefatos e resultados de teste de forma adequada, facilitando o reuso e a composição dos mesmos pelos clientes. Esta abordagem é complementada em [6] onde é apresentado um método de teste de integração funcional para sistemas baseados em componentes com a finalidade de testar o componente dentro do contexto onde ele será inserido, isto é, testar a forma como ele irá interagir com a aplicação e com os demais componentes que poderão compor o sistema.

Embora estes métodos tenham sido concebidos de forma a potencializar a sua automação, ferramentas de suporte ainda não foram desenvolvidas. Dessa forma, todas as suas etapas precisam ser manualmente executadas, o que torna esse processo bastante oneroso. Além disso algumas das características destes métodos precisam ser melhor definidas e/ou adaptadas para permitir a sua automação.

2 Trabalhos Relacionados

Esta seção descreve suscintamente alguns dos principais trabalhos desenvolvidos na área de automação de teste funcional que foram estudados durante a pesquisa bibliográfica do presente trabalho.

Em [1], é apresentada a técnica TOTEM (*Testing Object-orienTed systEms with the unified Modeling language*) que propõe um meio de derivação de casos de teste a partir de artefatos de análise de um sistema orientado a objetos especificados em UML (Diagramas de Casos de Uso, Classe e Seqüência). Esta técnica foi utilizada na definição do método de teste proposto em [4].

A derivação de casos de teste a partir de especificações sob a forma de diagramas de estado UML é o objetivo da ferramenta UMLTest apresentada em [8]. Entretanto, como concluído pelos próprios autores, nem todas as informações importantes de um sistema podem ser obtidas a partir de diagramas de estado, o que limita a aplicabilidade desta ferramenta. Uma abordagem semelhante pode ser vista em [5] onde casos de teste são gerados a partir de casos de uso de um sistema. Através desta técnica, um caso de uso é convertido em um diagrama de estados UML e este, por sua vez, em casos de teste. Esta técnica foi parcialmente automatizada sendo necessária a realização manual de algumas de suas fases.

Também temos em [2] uma abordagem para geração automática de teste a partir de modelos UML. O modelo UML utilizado é uma coleção de diagramas de classe, estado e objetos. Este trabalho apresenta características bem interessantes, a exemplo da utilização de modelos em formato XMI (*XML Metadata*

Interchange) e de geração de especificações IF (*Intermediate Format*) que podem ser aplicadas a um verificador de modelos ou à ferramenta TGV (*Test Generation with Verification*).

Trabalhos interessantes também vêm sendo realizados pelos pesquisadores do projeto AGEDIS[3] (*Automated Generation and Execution of Test Suites for DIstributed Component-based Software*), um grande projeto de pesquisa europeu que vem desenvolvendo metodologias e ferramentas para automação de teste de software com ênfase em sistemas baseados em componentes distribuídos.

A automação também está presente [10] onde foi desenvolvida a ferramenta de teste TORX que, a partir de especificações LTS (*Labelled Transition Systems*), provê geração automática, implementação, execução e análise de testes de maneira on-the-fly. Essa ferramenta possui uma arquitetura flexível e modular com interfaces bem definidas, o que lhe permite ser facilmente expandida para trabalhar com outras linguagens de especificação.

Como é possível observar, especificações UML vêm sendo utilizadas na maioria destes trabalhos. No entanto, à exceção de [1], todos se limitam a utilização direta ou indireta de diagramas de estado, pouco utilizados se comparados a diagramas de classe e seqüência. Nota-se também uma preocupação no desenvolvimento de ferramentas compatíveis com o formato XMI e com arquiteturas flexíveis, entretanto estas características ainda estão sendo pouco utilizadas.

3 Fundamentação Teórica

O presente trabalho tem como base os métodos propostos em [4] e [6]. Estes métodos além de combináveis, apresentam a vantagem de cobrir todo o processo de teste e fazer uso de especificações (artefatos) usuais. No primeiro é proposto um método de teste funcional para verificação de componentes que tem como objetivo a geração de casos de teste a partir de especificações UML e restrições OCL de componentes de software. Entre os artefatos UML utilizados estão:

- *Diagrama de Classes* onde são descritas, de forma estrutural, o modelo conceitual, as classes do sistema assim como as associações entre elas.
- *Restrições OCL* que especificam os contratos (pré e pós-condições) das operações das interfaces do componente.
- *Diagrama de Casos de Uso* que representam as funcionalidades do sistema.
- *Diagramas de Seqüência* que especificam os cenários de uso das funcionalidades do sistema.

O método, que aborda a perspectiva de teste de componentes voltado para o fornecedor, é composto por três fases:

Planejamento dos Testes. Esta etapa é iniciada logo após a definição dos requisitos e faz uso de artefatos de análise como o modelo conceitual e diagramas de casos de uso. No planejamento são tomadas decisões como que funcionalidades serão testadas e o quanto cada funcionalidade será testada. Para tanto são usadas técnicas como a análise de riscos e criticalidade.

Especificação dos Testes. Nesta etapa são gerados os modelos de teste a partir dos quais são derivados os casos de teste, dados e oráculos. Ela é composta por três sub-etapas a saber: Seleção de Casos de Teste, Geração de Oráculos de Teste e Seleção de Dados de Teste.

Construção, Execução e Verificação de Resultados. Nesta etapa, as informações geradas na fase anterior são usadas para implementar os casos de teste e oráculos. Os casos de teste e oráculos para cada caso de uso são implementados através de uma classe de teste.

O método ainda determina que os artefatos de teste gerados para o componente sejam empacotados e disponibilizados junto com o mesmo.

[3] http://www.agedis.de

Em [6], é proposto uma extensão do método anterior para a realização de teste de integração de vários componentes durante a composição de uma aplicação. Esse trabalho, por sua vez, aborda a perspectiva de teste de componentes voltado para o cliente, ou seja, o esforço de teste concentra-se na verificação das interfaces entre os componentes de uma aplicação a fim de garantir que a sua combinação produz um comportamento esperado. Além dos artefatos UML utilizados pelo método anterior este método faz uso ainda de Diagramas de Colaboração para especificar as interações existentes entre os componentes e baseia-se no uso de um diagrama de dependências de teste como proposto em [9].

4 Objetivos e Metodologia do Trabalho

O objetivo deste trabalho é a automação dos métodos de teste propostos em [4] e [6]. Nesse sentido, algumas características e/ou algoritmos destes métodos precisam ser melhor definidos de forma a possibilitar a sua automação, a exemplo da utilização de técnicas mais eficientes de seleção de dados e do formato de empacotamento a ser utilizado para os artefatos de teste. Outro ponto importante é flexibilizar os métodos de tal forma que eles possam trabalhar com especificações UML aliada a outros formalismos além de OCL, como, por exemplo, Object Z.

Para tanto, está sendo desenvolvida a ferramenta de suporte SPACES (*SPecification bAsed Component tESter*). Entre as características gerais desta ferramenta destacam-se:

- Integração com ferramentas de modelagem UML que exportem para o formato XMI, padrão da OMG[4] para troca de informações;
- Possibilidade de utilização de diferentes linguagens de especificação de restrições (OCL, Object Z, etc.);
- Geração de código de teste para diferentes linguagens de programação e/ou plataformas;
- Possibilidade de re-execução dos casos de teste gerados para novos dados fornecidos dinâmicamente pelos usuários.

Com o intuito de se atingir o objetivo proposto para o trabalho, foram definidas as seguintes metas:

1- Análise do Estado da Arte em automação de teste funcional. Consistindo na realização de uma pesquisa bibliográfica sobre trabalhos relacionados a automação de teste funcional para software orientado a objetos e/ou baseado em componentes.

2- Análise dos métodos propostos em [4] e [6]. Com o objetivo de compreender cada uma das etapas destes métodos, identificando, principalmente, as partes automatizáveis e propondo soluções para alcançar este objetivo.

3- Estudo das principais ferramentas utilizadas para teste funcional. Com o objetivo de identificar padrões e técnicas bem estabelecidas, assim como fazer um levantamento e estudo das tecnologias e ferramentas necessárias à concepção do trabalho.

4- Projeto e implementação da ferramenta SPACES. Após a definição da arquitetura, o projeto da ferramenta será dividido em duas fases. A primeira endereçará apenas as funcionalidades relacionadas com o teste funcional de componentes (perspectiva do fornecedor), enquanto que a segunda acrescentará as funcionalidades necessárias para o teste de integração funcional de componentes (perspectiva do cliente).

5- Definição e aplicação do estudo de caso. Durante o desenvolvimento da ferramenta, será feita a escolha e utilização de uma aplicação como estudo de caso, de forma que a mesma possa contribuir na concepção do projeto, fornecendo um *feedback* realista para o trabalho.

[4] *Object Management Group*. http://www.omg.org

5 Resultados Esperados

Ao final do trabalho, a ferramenta SPACES deverá ser capaz de proceder a geração, execução e análise de casos de teste de forma automática a partir de especificações UML e restrições OCL gerando código de teste em Java segundo o framework JUnit. Entretanto, será possível extendê-la de maneira que as restrições do modelo possam ser expressas em outros formalismos e o código de teste possa ser gerado em outras linguagens de programação.

Espera-se que a automação dos métodos propostos em [4] e [6] possa maximizar a sua chance de utilização em aplicações mais complexas uma vez que a execução manual de suas atividades além de ser um processo oneroso, pode resultar na inserção de erros comprometendo, assim, o resultado dos testes.

Além disso, espera-se que o desenvolvimento de uma ferramenta de geração de teste a partir de especificações UML contribua para que desenvolvedores e empresas não habituados a outras especificações menos usuais, tenham acesso a mecanismos eficientes para melhorar a qualidade de seus produtos de desenvolvimento.

6 Estado Atual do Trabalho

Como previsto na meta 1, foi realizada a pesquisa bibliográfica através da qual foi possível identificar as principais iniciativas no que se refere a automação de teste funcional. De posse desse conhecimento, foi realizado o estudo dos métodos propostos em [4] e [6], que resultou em algumas propostas de adaptação destes métodos de forma a possibilitar a sua automação (meta 2). Um exemplo destas adaptações é a geração de modelos de teste diretamente dos diagramas de seqüência, atividade esta que anteriormente era realizada de forma manual e sistemática. Em seguida, foi realizado um levantamento e estudo das tecnologias e ferramentas necessárias para a concepção do trabalho. Esse estudo serviu de base para o projeto da ferramenta SPACES.

O projeto arquitetural da ferramenta foi elaborado levando-se em consideração as funcionalidades que a mesma precisa prover no sentido de automatizar ao máximo as atividades dos métodos propostos. Assim, estas funcionalidades foram agrupadas em módulos correspondentes às fases do processo. A Figura 1 apresenta a arquitetura proposta para a ferramenta. O padrão XMI foi adotado para permitir a utilização de especificações construídas nas principais ferramentas case. Desta forma, torna-se possível a integração da ferramenta de teste com ferramentas de modelagem UML que exportem para esse formato, a exemplo de *IBM Rational Rose*[5] e *Gentleware Poseidon for UML*[6].

De uma forma geral, o funcionamento da ferramenta pode ser descrito como a seguir. Inicialmente, a especificação UML, em formato XMI, é processada pelo *parser* que instancia objetos correspondentes às informações do modelo UML (Diagramas, Classes, Operações, Constraints, etc.). Essas informações são processadas pelos módulos Seletor de Casos de Teste, Gerador de Oráculo e Seletor de Dados de Teste. As informações obtidas (Casos de Teste, Oráculos e Dados) são então utilizadas pelo módulo Gerador do Código de Teste. Neste momento é escolhida a linguagem de programação na qual serão geradas as classes de teste. O código resultante deverá estar de acordo com algum framework de teste da família XUnit (JUnit, CPPUnit, etc.) e com a linguagem de programação utilizada na implementação sob teste. Ao final do processo, o código de teste é empacotado sob a forma de um componente de teste que é então distribuído juntamente com o sistema, possibilitando a execução dos casos de teste com dados selecionados e/ou fornecidos pelo usuário no momento da execução.

Atualmente, o desenvolvimento da ferramenta SPACES encontra-se em sua primeira fase, tendo sido concluída a implementação dos módulos Parser XMI e Seletor de Casos de Teste.

[5] http://www.rational.com

[6] http://www.gentleware.com

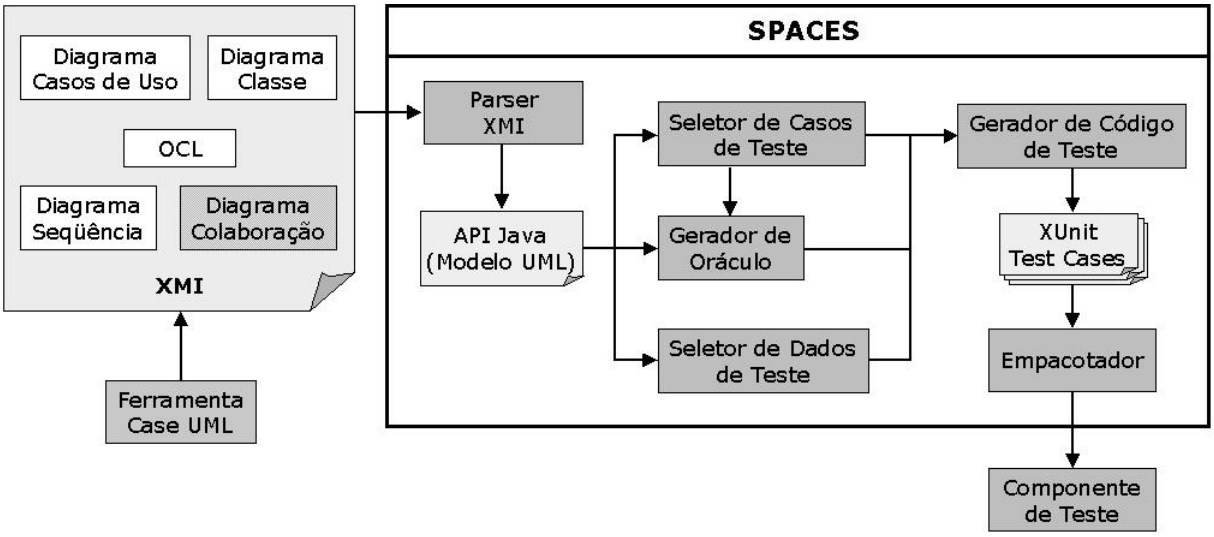

Figura 1: Arquitetura da Ferramenta SPACES

Referências

[1] Lionel Briand and Yvan Labiche. A UML-based approach to system testing. *Lecture Notes in Computer Science*, 2185:60–70, 2001.

[2] A. Cavarra, C. Crichton, J. Davies, A. Hartman, T. Jeron, and L. Mounier. Using uml for automatic test generation. In *Proceedings of TACAS 2002*, 2001.

[3] MH. Chen, Y. Wu, and J. Offutt. Uml-based integration testing for component-based software. In *Proceedings of the 2nd International Conference on COTS-Based Software Systems (ICCBSS), February 2003*, 2003.

[4] Carina Machado de Farias and Patrícia D. L. Machado. Um método de teste funcional para verificação de componentes. *XVII Simpósio Brasileiro de Engenharia de Software*, 2003.

[5] P. Fröhlich and J. Link. Automated test case generation from dynamic models. In *Proceedings of ECOOP 2000, LNCS 1850*, pages 472–491. Springer, 2000.

[6] Cidinha Costa Gouveia. Um método de teste de integração para sistemas baseados em componentes. *Dissertação de mestrado - COPIN, UFCG*, 2004.

[7] E. Martins, C. Toyota, and R. Yanagawa. Constructing self-testable software components. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (DSN '01), Washington - Brussels - Tokyo*, pages 151–160. IEEE, 2001.

[8] Jeff Offutt and Aynur Abdurazik. Generating tests from UML specifications. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723 of *LNCS*, pages 416–429. Springer, 1999.

[9] Y. L. Traon, T. Jéron, JM. Jézéquel, and P. Morel. Efficient object-oriented integration and regression testing. *IEEE Transactions on Reability*, 49(1), March 2000.

[10] J. Tretmans and E. Brinksma. Côte de resyste - automated model based testing. In M. Schweizer, editor, *Proceedings of Progress 2002 - 3rd Workshop on Embedded Systems, STW Technology Foundation, Utrecht, The Netherlands*, pages 246–255, 2002.