PIBIC/CNPq/UFCG-2009

ESTENDENDO O PERFIL DE TESTES U2TP PARA DAR SUPORTE AO TESTE DE SOFTWARE BASEADO EM COMPONENTES

Everton L. G. Alves[1], Patrícia D. L. Machado[2]

RESUMO

Este trabalho teve o objetivo de analisar e propor melhorias para a metodologia de testes em sistemas baseados em componentes BIT, bem como aplicar ao perfil de testes da UML (U2TP) as extenções necessárias para que este possa tratar o teste de componentes. Em adição, este trabalho busca validar, via experimetação com usuários reais, o uso da ferramenta de geração automática de artefatos de testes para componentes, MoBIT.

Palavras-chave: teste de componente, automação, U2TP

EXTEND THE UML TESTING PROFILE TO SUPPORT SOFTWARE TESTING COMPONENT-BASED

ABSTRACT

This work had the objective of investigating and proposing improvements to the methodology of testing in component-bases systems named BIT and also apply to the UML testing profile (U2TP) extensions to enable it the test components. In addition, this paper validated, via experimentation with real users, the use of the MoBIT tool.

Keywords: component testing, automation, U2TP

INTRODUÇÃO

O desenvolvimento de sistemas computacionais tem se especializado com os passar dos tempos. Erros cometidos em projetos passados são objetos de estudo da comunidade científica para buscar melhorias tanto no processo de desenvolvimento dos softwares, bem como para desenvolver nossas técnicas e ferramentas que auxiliem esta atividade.

Dentre as práticas difundidas para melhoria da qualidade do software, uma das mais promissoras é o chamado desenvolvimento de Software Baseado em Componentes (SBC). Essa prática tem como idéia central que módulos do sistema (componentes) sejam desenvolvidos e testados independentemente para que, em seguida, seja realizado o acoplamento dos mesmos através de suas interfaces (de requisição e de provisão). Tais características ajudam que conceitos importantes para um bom software, tais como, encapsulamento e a localidade (SZYPERSKI, 1998), estejam presentes no produto final. Diversos resultados satisfatórios têm sido alcançados com o emprego dessa prática na indústria e academia.

Como os componentes geralmente são desenvolvidos para serem reusados em diferentes plataformas, é muito importante que seus testes sejam facilmente modificáveis para que reflitam o ambiente onde o componente está inserido. Para isto tem crescido a utilização de SBC combinado com Testes Baseados em Modelos – MBT (GROSS, 2003) (EL-FAR & WHITTAKER, 2001). Com MBT, conjuntos de casos de teste podem ser automaticamente gerados à medida que existam os modelos de especificação do software,

[1] Aluno do Curso de Ciência da Computação, Centro de Engenharia Elétrica e Informática, UFCG, Campina Gande , PB, E-mail: everton@dsc.ufcg.edu.br

[2] Ciência da Computação, Profª. Doutora, Centro de Engenharia Elétrica e Informática, UFCG, Campina Gande , PB, E-mail: patricia@dsc.ufcg.edu.br

favorecendo o reuso de artefatos de desenvolvimento e formando um forte elo entre as equipes de desenvolvimento e de teste.

Paralelamente a SBC uma das abordagens recentes voltada ao desenvolvimento de software com alto padrão de qualidade é o Desenvolvimento Dirigido por Modelos (MDD) (STAHL et al., 2006). Essa abordagem objetiva focar o desenvolvimento de software nos modelos e transformações entre modelos, e não mais em linguagens de programação. Com essa mudança de foco, a equipe de desenvolvimento concentrará seus esforços na construção de modelos mais completos e precisos, viabilizando a total geração de código de forma automática. Uma das realizações de MDD que têm obtido resultados mais concretos e promissores é MDA - *Model Driven Architecture* (KLEPPE et al., 2003). MDA foi à realização de MDD lançada pela OMG - *Object Management Group* (OMG, 2006) e, por isso, tem como base para sua realização os padrões propostos por essa organização.

Cada nova abordagem tem suas vantagens. Logo, o objetivo dos desenvolvedores é sempre combinar os pontos positivos de cada uma de maneira a garantir uma maior qualidade no software final, porém de uma forma que não cause maior sobrecarga.

O objetivo desse projeto de iniciação científica é: investigar a aplicação prática do teste embutido de contrato applicado a componentes de software, experimentar a ferramenta MoBIT (LIMA et al, 2007) de maneira a localizar seus pontos falhos e investigar o uso do perfil de testes U2TP para a especificação e projeto de teste para SBC estendendo-o para dar suporte ao BIT dentro de um processo de MDT.

MDA

Model Driven Architecture (KLEPPE et al., 2003) é uma realização de MDD proposta pela OMG que tem se popularizado. Esta tem como características ser uma abordagem padronizada aberta e independente de fornecedor. O principal objetivo da OMG ao propor a abordagem MDA foi padronizar diversos modelos para que as empresas os possam usar para representar aplicações. Alguns desses modelos são independentes dos detalhes de implementação (CIM - *Computational Independent Model* e PIM - *Platform Independent Model*), outros contemplam características de linguagens de programação (PSM – *Platform Specific Model*). Assim, fazendo uso da MDA uma empresa será capaz de construir um sistema independente da tecnologia de implementação existente da época, e caso a posteriori haja a necessidade de uma mudança de plataforma, essa passagem poderá ser feita mais facilmente, pois apenas os modelos específicos de plataforma (PSM) sofreriam alterações. Logo, arquiteturas poderão servir por décadas a empresas, muito diferente do que acontece atualmente.

A idéia que MDA apregoa é que todo o processo deva ser realizado de forma automática. Para isso, indica o uso de regras de transformação entre modelos. As regras podem ser construídas tanto para refinamento de modelos de uma mesma categoria (PIM para PIM, por exemplo), quanto para mudança de categorias de modelos (PIM para PSM, por exemplo). Dentre os padrões propostos pela OMG, o que é utilizado amplamente em MDA é a UML – *Unified Modeling Language* (OMG, 2007). Outros padrões propostos pela OMG para trabalhar com modelos são: XMI, OCL e MOF.

MDT

A derivação dos casos de teste a partir dos modelos da especificação funcional do sistema é comumente chamado de Teste Baseado em Modelos (MBT) (EL-FAR & WHITTAKER, 2001).

A geração automática dos testes a partir de modelos favorece o reuso dos artefatos de desenvolvimento e ajuda a formar um elo entre a equipe de desenvolvimento e a de testes. Teste Dirigido por Modelos (MDT) (HECKEL &LOHMANN, 2003) (BACKER et al. ,2007) é uma abordagem de MBT que faz uso de práticas de MDD para a geração automática de artefatos de teste (casos de teste, oráculos etc) de acordo com regras de transformação pré-definidas, possivelmente a partir de modelos de desenvolvimento. Esses artefatos gerados poderão servir para a execução em diferentes plataformas. Dentre as vantagens da utilização de MDT em relação à MBT a principal é que, em MBT, os casos de teste são derivados a partir dos modelos de desenvolvimento fracamente conectados, onde estes são normalmente incompletos de informações necessárias para os testes (restrições, casos alternativos, dentre outros). Com a utilização das práticas de MDD, onde modelos são o centro do desenvolvimento, tais informações poderão ser naturalmente incorporadas.

UML/U2TP

A UML (UML, 2007) – *Unified Modeling Language* – é a linguagem de modelagem gráfica que se tornou padrão entre empresas e academia para modelagem de sistemas computacionais, isto devido a sua simplicidade de expressão e permissibilidade de representação de praticamente todos os conceitos de um sistema. A UML também é sustentada pela OMG e por isso é padrão para diversas abordagens lançadas por essa organização.

A UML atualmente está na sua versão 2.0, possuindo treze diferentes diagramas, tornando possível a modelagem de todas as etapas do desenvolvimento do software, cada qual possuindo um ou mais diagramas correspondentes.

UML provê mecanismos - perfis - para extensão de seus diagramas com o objetivo de adaptá-los as semânticas de domínios específicos. No contexto de testes, a OMG definiu um perfil de testes para a versão 2.0 da UML - *UML 2.0 Test Profile* (U2TP)(OMG, 2005) a fim de facilitar o projeto, visualização, especificação, análise, construção e documentação de artefatos de teste funcional. U2TP cobre amplamente os diferentes conceitos que são abordados em um projeto de teste.

Esse perfil foi definido com base no meta-modelo de UML 2.0, visando o reuso de conceitos já existentes, propiciando a integração de MDD e MDT. O perfil é organizado em quatro grupos de conceito: i) Arquitetura de Teste (*Test Architecture*); ii) Comportamento de Teste (*Test Behaviour*); iii) Dados de Teste (*Test Data*) e iv) Conceitos de Tempo (*Time Concepts*). Cada grupo elenca um conjunto de conceitos que especificam elementos importantes para a construção e execução dos artefatos de teste, e a utilização conjunta leva a uma especificação mais completa em se tratando de testes.

BIT

Tratando-se de componentes, uma das verificações mais importantes a ser feita é aquela que testa se a comunicação entre um componente que possui o papel de servidor (fornecedor de serviços) e outro considerado cliente (consumidor dos serviços) esta sendo feita satisfatoriamente.

É sabido que, para se fazer uso de um componente, não é preciso que se tenha acesso a informações de como este foi construído, bastando apenas que suas interfaces de comunicação sejam conhecidas. Baseando-se nessa característica, os testes de componentes atuam na verificação se os contratos de cada componente estão sendo seguidos, ou seja, se o papel das interfaces de cada componente está sendo cumprido corretamente.

Seguindo essa idéia, (ATKINSON & GROSS, 2002) propuseram o BIT - *Built-In contract Testing*, método integrado com a metodologia de desenvolvimento de SBC KobrA (ATKINSON et al., 2002) onde são embutidos testes do comportamento dos servidores nas próprias interfaces. Utilizando estas interfaces, os clientes são capazes de verificar se os servidores cumprem seus contratos. Os contratos são basicamente as especificações dos componentes produzidos pela metodologia KobrA.

A vantagem da utilização dessa metodologia de testes é que, como o componente será dotado da capacidade de controlar sua execução, será possível testar a troca cliente-servidor em tempo de execução, reduzindo-se assim o esforço manual para verificação do sistema.

O BIT determina que os artefatos de especificação e realização dos componentes devem ser estendidos com os artefatos de teste, ou seja, aos modelos originários da utilização da metodologia KobrA, novas estruturas de testes devem ser adicionadas. Os artefatos adicionais são componentes de teste e interfaces inclusas no componente cliente. Esses componentes de teste são compostos por métodos que definem e dão suporte a execução dos casos de teste.

Para o componente servidor há a criação da interface de testes que consiste em um conjunto de operações onde, para cada estado da máquina de estados comportamentais, proveniente da modelagem do comportamento do componente (seguindo a metodologia KobrA), serão derivados duas operações. A primeira conduzirá o componente ao estado específico, e a segunda fará a verificação se o mesmo está ou não no estado em questão. Essa interface é acessível ao componente de teste, que é subclasse do servidor.

Outros componentes de teste também são derivados, cada qual desempenhando um papel específico, tanto para a configuração quanto para a execução dos casos de testes.

MATERIAL E MÉTODOS

Este trabalho foi desenvolvido no laboratório do Grupo de Métodos Formais – GMF, parte do Centro de Engenharia Elétrica e Informática - CEEI da Universidade Federal de Campina Grande – PB.

Os seguintes passos foram planejados para desenvolvimento do projeto em questão:

1. Realização de pesquisa para fundamentação teórica.
2. Proposição de complementos para o BIT e investigação do uso do U2TP para o teste de SBC.
3. Investigação de uma proposta de extensão ao U2TP.
4. Implementação de novos módulos na ferramenta MoBIT.
5. Realização de estudos de caso.
6. Apresentação de Resultados.

A seguir, será apresentada mais detalhadamente cada uma das etapas planejadas.

A primeira etapa tinha por objetivo que os executores do projeto pudessem realizar um levantamento criterioso na literatura especializada, buscando compreender os novos conceitos que seriam necessários para o desenvolvimento do projeto. Em especial, nesse primeiro passo, a maior parte do tempo deveria ser alocada para investigar as novidades presentes na UML2.0.

O segundo passo da metodologia seria dedicado a investigação de como usar a especificação do BIT de maneira prática. Tal atividade é necessária, pois apesar da especificação da metodologia de testes BIT ser bastante ampla e complexa, existem algumas lacunas a serem preenchidas para que o mesmo possa ser posto em uso. Ainda nesta etapa, parte do tempo seria destinado ao estudo do perfil de testes e verificação de como este poderia ser usado para tratar o teste de componentes.

Durante o terceiro passo um conjunto de adições ao perfil de testes U2TP deveria ser criado, onde estas adições seriam responsáveis por complementá-lo no tocante ao tratamento de testes de componentes.

Seguindo, na etapa 4, as possíveis modificações decorrentes da etapa 2, bem como as possíveis extensões no perfil de testes (etapa 3) seriam implementadas na ferramenta MoBIT. Tal atividade teria por objetivo estender a ferramenta para que a mesma possa gerar artefatos de teste mais completos, assim como possuir uma perspectiva de testes de componentes diferente da usada no BIT.

Finalizando o projeto, estudos de caso deveriam ser construídos para validar os resultados obtidos e os novos conceitos empregados (etapa 5) e por fim, relatórios e artigos seriam escritos para apresentação à comunidade científica os resultados do projeto (etapa 6).

RESULTADOS E DISCUSSÃO

Passada a etapa de embasamento teórico (etapa 1 da metodologia), tivemos como resultado da execução do passo 2 a sugestão de diversas modificações consideradas imprescindíveis na metodologia BIT. Estas modificações foram desenvolvidas para que a praticidade e automação, características essenciais para sua aplicação, pudessem existir. A seguir, são elencadas as diversas modificações propostas para o real uso do BIT na sua plenitude.

I. Inclusão de um elemento controlador ao qual chamamos de *Arbiter*, este possuirá uma interface composta por três operações: *setVerdict, getVerdict, definesPartialVerdict.* Essa inclusão vem da necessidade da composição de um resultado para a suíte de testes que pudesse ser analisado, bem como, que esse resultado possa ser composto dinamicamente assim como fazem os mais conhecidos *frameworks* de testes (*JUnit, PyUnit* etc).

II. Para auxiliar a diferenciação dos veredictos referentes à execução dos casos de teste, introduzimos uma enumeração (*Verdict*), onde os possíveis resultados elencados serão: i) *pass,* indicando que o teste (ou a suíte) foi executado sem problemas; ii) *fail,* indicando que os testes falharam devido a defeitos do sistema; iii) *inconclusive,* indicando que algo inesperado aconteceu, seja uma exceção lançada ou alguma pré-condição que deveria ser satisfeita para a execução dos testes não foi configurada corretamente; e iv) *error*, indicando que um erro ocorreu durante a execução dos testes.

III. É sabido que o elemento *Testable* do componente servidor possui uma interface de testes composta por duplas de operações para cada estado da máquina de estados, e que essas operações devem ser implementadas pela equipe de testes para que a suíte possa ser executada. Uma dessas operações é a "*setTo + nome do estado",* que é usada para forçar o componente a estar num determinado estado. Essa operação é muito importante, pois permite que um caso de testes possa começar de outro estado que não o inicial. A modificação realizada nessa operação foi à inclusão de um novo parâmetro para diferenciar como deve estar o componente no determinado instante, pois é possível que um componente esteja num mesmo estado em situações diferentes, e estas situações devam ser testadas também diferentemente. Essa necessidade fica clara quando, nas máquinas de estado, tratamos de transições que chegam a um estado cada qual com guardas diferentes.

IV. Outro problema que existe quando desejamos fazer a geração automática dos artefatos de teste diretamente dos modelos de desenvolvimento é a parametrização das chamadas das operações nos testes. A dificuldade em saber quais valores passar para que os testes possam ser executados é ainda maior se a única fonte de dados forem os modelos. A metodologia BIT nada indica para resolver esse tipo de problema. A solução que desenvolvemos para tal situação foi delegar a equipe que for executar os testes a tarefa de definir quais valores os parâmetros devem assumir, pois somente a equipe possuirá conhecimento suficiente da aplicação e saberá como devem ser os parâmetros em cada situação. Continuando na idéia de automatização de código pregada pelo BIT, propomos que todo e qualquer parâmetro, objeto ou variável seja pré-inicializada com um valor *default* que posteriormente possa ser modificada pelo testador. Essa pré-inicialização deve estar presente no *Tester* do componente cliente.

A investigação que levou a proposição das modificações listadas anteriormente trouxe como resultado secundário uma melhor compreensão das necessidades existentes no desenvolvimento de elementos necessários para a atividade de testes de componentes. Tal experiência foi bastante relevante durante a realização das etapas posteriores do projeto.

Em sequência, decidimos adiar a execução do passo 3 da metodologia e passarmos diretamente para o passo 4. Esse salto foi decidido, pois antes da definição das extensões necessárias para o perfil U2TP notou-se a necessidade de uma melhor validação se as novas ideias desenvolvidas para o BIT eram realmente relevantes no contexto do desenvolvimento baseado em componentes. Para tal, durante no passo 4 executou-se a implementação dos elementos presentes no BIT (completos) na ferramenta de geração automática de artefatos de testes para componentes MoBIT (LIMA et al, 2007).

Um conjunto de regras de transformação escritas na linguagem ATL (exemplificada na **Figura 1**) foram incorporadas ao código já existente da ferramenta MoBIT. Através dessas regras, tornou-se possível gerar automaticamente, a partir dos modelos de desenvolvimento, todas as estruturas necessárias para a execução dos testes, tanto em um nível de abstração independente de plataforma quando especificamente para a plataforma Java (plataforma empregada na ferramenta). Com a implementação realizada durante o passo 4, obteve-se como resultado um código de testes (inclusive os casos de teste), que pode ser diretamente executado sem grandes modificações. Portanto, conforme indica MBT/MDT, com MoBIT já é possível criar e executar toda a estrutura para teste de componentes BIT automaticamente usando somente os modelos de aplicação como entrada.

```
rule createArbiter (model: uml2!Model){
  to arb:uml2!Class(
     name <- 'Arbiter',
     ownedAttribute<- Set{ver1, partialVerd},
     ownedOperation <- Set{ooArbiter1, ooArbiter2, ooArbiter3}  ),

  ver1: uml2!Property(
     name<- 'verdict',
     type<- thisModule.verdict  ),

  partialVerd: uml2!Property(
     name<- 'partialVerdict',
     type<- thisModule.verdict  ),

  ooArbiter1:uml2!Operation(
     name <- 'getVerdict',
     ownedParameter <- paramGetVeredict,
     visibility <- #public  ),

  paramGetVeredict:uml2!Parameter(
     type <- thisModule.verdict,
     name <- 'v',
     direction <- #return   ),

  ooArbiter2:uml2!Operation(
     name <- 'setVerdict',
     visibility <- #public,
     ownedParameter <- paramSetVeredict  ),
```

Figura 1: Parte da regra de transformação que cria o elemento *Arbiter*.

Em sequência, a etapa 5 foi destinada a realização de estudos de caso que pretendiam averiguar experimentalmente um conjunto de tópicos: i) analisar se a ferramenta MoBIT estava trabalhando adequadamente; ii) averiguar se os novos conceitos implementados estavam coerentes; iii) descobrir como testadores não treinados conseguiriam usar a ferramenta e as metodologia de testes; e iv) analisar se testadores conseguem usar o perfil de testes para implementar uma arquitetura de testes adequada sem maiores problemas.

Os estudos de caso foram realizados da seguinte maneira: com a ajuda da turma da disciplina de “Teste Dirigido por Modelos”, oferecida pelo programa de pós-graduação em Informática da UFCG, três grupos de alunos foram formados onde cada grupo ficou responsável por: i) escolher um sistema de software baseado em componentes; ii) usando a ferramenta MoBIT, gerar os artefatos de testes para suas respectivas aplicações; e iii) usando o perfil de testes da UML (U2TP) tentar, independentemente dos artefatos gerados anteriormente, projetar da melhor maneira possível a atividade de testes dos seus respectivos sistemas.

Para análise dos resultados, cada grupo apresentou dois importantes produtos: os artefatos obtidos (modelos e código gerados) pela ferramenta e as métricas colhidas no decorrer do projeto. As métricas coletadas forma essencialmente as seguintes: tempo gasto para realização da atividade e número de defeitos encontrados. A tabela da **Figura 2** mostra as métricas colhidas por um dos grupos durante a realização da primeira fase do projeto.

Etapa	Defeitos	Tempo
Modelagem	18	10:55
Uso de Mobit	4	8:06
Total	22	**19:01**

Figura 2: Compilação das métricas coletadas por uma das equipes realizadoras dos estudos de caso.

A **Figura 3** mostra o resumo geral das métricas colhidas por todos os grupos ao final da realização da atividade. A partir destas métricas, das apresentações, bem como do acompanhamento da realização da atividade, foi possível fazer as seguintes conclusões da realização dos estudos de caso:

- Para uso em sua plenitude da ferramenta MoBIT e dos conceitos totais de MDT, é necessário que o usuário da ferramenta possua certa experiência em testes, UML e metodologias de teste.
- A ferramenta MoBIT atualmente está com documentação e interface fraca, o que dificulta o seu uso por usuários não treinados.
- Os casos de teste gerados pela ferramenta foram válidos e de boa qualidade, quase em sua totalidade.
- A maioria dos defeitos encontrados foram provenientes de erros da modelagem do sistema, o que nos leva a inferir que existe a necessidade de que os envolvidos entendam melhor a importância que uma modelagem completa assume em MDT.
- A maioria do tempo gasto pelas equipes foi na tentativa de descobrir a origem dos erros atestados pela ferramenta.
- Com o auxílio de MoBIT foram localizados tanto erros de modelagem quanto erros de requisitos do sistema que haviam passados despercebidos.
- A especificação do U2TP foi considerada confusa, levando a realizações indevidas em certas circunstancias.
- Os grupos consideraram o perfil mais completo no que diz respeito à modelagem de situações e estruturas que não são facilmente modeladas no BIT.
- A utilização do U2TP foi dificultada por não existir nenhuma ferramenta de auxílio, automatizando o processo.

Etapa	Defeitos	Tempo
Modelagem	88	20:00
Uso de MoBIT	8	10:40
Aplicação U2TP	41	36:30
Total	**137**	**67:10**

Figura 3: Compilação geral das métricas coletadas.

CONCLUSÕES

Apesar da não finalização do projeto como planejado (a bolsa foi interrompida antes do período previsto devido à entrada do bolsista no programa de mestrado da instituição), com a extensão do perfil U2TP, acreditamos que os resultados alcançados podem ser considerados de grande relevância, pois diversos avanços na área de testes de componentes foram alcançados, dentre eles: i) a adição de complementos à metodologia e ao perfil do BIT, tornando-o mais completo e possibilitando o seu uso de maneira automática; ii) o aprimoramento da ferramenta MoBIT, incluindo novos elementos que tornam seus resultados mais completos e úteis para a utilização prática da mesma; e iii) realização de experimentos com usuários reais que apontou as dificuldades tanto do uso da ferramenta MoBIT como do uso de MDT como um todo.

Como trabalhos futuros para continuação da pesquisa, podemos enxergar as seguintes possibilidades: i) investigar a extensão do perfil de testes com as ideias necessárias para o teste de componentes; ii) implementar, junto a MoBIT, a aplicação automática do perfil de testes U2TP; iii) tratar as deficiências de MoBIT levantadas durante a realização dos estudos de caso; e iv) investigar como as dificuldades encontradas para aplicação de MDT poderiam ser superadas.

AGRADECIMENTOS

Ao CNPq pela bolsa de Iniciação Científica, a professora Patrícia Machado pela orientação e importante colaboração. A estes, agradeço o apoio que contribuiu para meu aprendizado e aperfeiçoamento das minhas habilidades de pesquisador.

REFERÊNCIAS BIBLIOGRÁFICAS

ATKINSON, C. and GROSS, H.-G. Built-in contract testing in model-driven, component-based development. *Proceedings of ICSR - Workshop on Component-Based Development Processes.* 2002.

ATKINSON, C., BAYER, J., BUNSE, C., KAMSTIES, E., LAITENBERGER,O., LAQUA, R., MUTHIG, D., PAECH, B., WST, J., and ZETTEL, J. **Component-basedProduct Line Engineering with UML**. Component Software Series. Addison-Wesley. 2002.

BECK, K. Test-Driven Development by Example, Addison Wesley, 2003

GROSS, H. **Testing and the UML – A perfect fit, Technical Report**, Fraunhofer IESE, Report 110.03E

EL-FAR, I. K. and Whittaker, J. A. (2001). Model-based software testing. Encyclopedia on Software Engineering.

HECKEL, R. and LOHMANN, M. Towards model driven testing. *Electronic Notes in Theoretical Computer Science.* 2003, Vol. 82, 6.

KLEPPE, A., WARMER, J., and BAST, W. **MDA explained (The model-driven architecture:**

practice and promise). Object-Technology Series. Addison-Wesley, 2003.

LIMA, H., RAMALHO, F., MACHADO, P., and ALVES, E. L. G. Automatic generation of platform independent built-in contract testing. In Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software 2007.

OMG. *UML testing profile.* 2005. Technical Report. http://www.omg.org/cgibin/doc?formal/05-07-07. formal/05-07-07.

OMG, 2007. UML superstructure, v2.1.1. Technical Report formal/07-02-05, OMG. http://www.omg.org/cgi-bin/doc?formal/07-02-05.

STAHL, T., Voelter, M. and Czarnecki, K**.** *Model-Driven Software Development: Technology, Engineering, Management.* s.l. : Wiley, 2006.

SZYPERSKI, C. *Component Software: Beyond Object-Oriented Programming.* s.l. : Addison-Wesley, 1998.