Diretrizes para Melhoria da Qualidade dos Sistemas de Informação

Rogéria Cristiane Gratão de Souza [1], Selma Shin Shimizu Melnikoff [2]

[1] UNESP – Universidade Estadual Paulista (Apoio Fundunesp – Proc. 01428/04)
Rua Cristovão Colombo, 2265 – 15.054-000 – São José do Rio Preto – SP – Brasil

[2] EPUSP - Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, trav. 3, no. 158 – 05.508-900 – São Paulo – SP – Brasil

rogeria@ibilce.unesp.br, selma.melnikoff@poli.usp.br

Abstract. *This article presents a guideline to improve the quality of software systems in order to produce suitable information. So, it is important to systematize the test activities since the earlier phases of the development cycle. As a result, this article consider the diagrams elaborated during the Analysis phase and identify the important information to the tests activities contributing to produce a reliable information system that meets the business requirements.*

Resumo. *Este artigo apresenta um conjunto de diretrizes, visando melhorar a qualidade dos sistemas de software para que produzam as informações adequadas. Para tanto, considera-se relevante sistematizar as atividades de teste para que sejam previstas desde as fases iniciais do processo de desenvolvimento. Assim, este artigo considera os diagramas elaborados na fase de Análise, identificando as informações relevantes para as atividades de teste e contribuindo para a obtenção de sistemas de informação confiáveis que atendem às reais necessidades da empresa.*

1. Introdução

Os recursos financeiros despendidos pelas organizações com Tecnologia da Informação (TI) são muito significativos para a maioria dos segmentos de empresas. Diante disso, Carr (2003) afirma que o ideal para as empresas é investir na melhoria da qualidade dos sistemas de software, com o intuito de reduzir os riscos de interrupção dos serviços dos seus atuais ambientes computacionais. Sendo assim, é necessário que as atividades relacionadas com o teste de tais sistemas sejam planejadas e executadas de forma sistemática, para que a maior parte dos erros sejam detectados e corrigidos antes que o sistema seja implantado. Ressalta-se que, caso os artefatos gerados durante o desenvolvimento apresentem informações relevantes para a elaboração dos documentos de teste e para a execução dos testes, o desenvolvimento como um todo ficará com qualidade melhor. Isso certamente contribuirá para o fornecimento de informações adequadas pelos sistemas de software desenvolvidos.

Diante da grande utilização do paradigma orientado a objetos (OO) para o desenvolvimento de sistemas, este artigo se baseia na tese de doutorado Souza(2003) que apresenta a definição das características de testabilidade nos modelos especificados em *Unified Modeling Language* – UML (Booch et al.(1999)).

Neste contexto, o objetivo deste artigo é definir diretrizes para o planejamento e execução das atividades de teste, considerando que os diagramas do Modelo de Análise da UML podem fornecer informações para apoiar estas atividades. Com isso, pode-se preparar o sistema para testes desde o início do processo de desenvolvimento, tornando os testes mais previsíveis e melhorando a qualidade das informações geradas.

A Seção 2 descreve alguns trabalhos relacionados, enfatizando a importância da UML para as atividades de teste. A Seção 3 apresenta as diretrizes definidas sobre uma estratégia de teste para sistemas de software OO. A Seção 4 aponta os resultados obtidos. A Seção 5 relata as conclusões e trabalhos futuros.

2. Contribuições da UML para as Atividades de Teste

Pesquisas mostram que os diagramas do Modelo de Análise fornecem informações para a geração automática de casos de testes, o que contribui para a garantia de realização de testes e, conseqüentemente, para a melhoria da qualidade do sistema de software obtido. Neste trabalho, considera-se que os Diagramas de Casos de Uso, Classes, Objetos, Seqüência, Colaboração, Estados e Atividades constituem uma boa base para descrever os artefatos da fase de Análise. Cabe ressaltar que os Diagramas de Objetos e Colaboração não são considerados neste artigo uma vez que suas contribuições para as atividades de teste também podem ser extraídas a partir dos Diagramas de Classes e Seqüência (Booch et al.(1999), Souza(2003)). Os diagramas elaborados na fase de Projeto (Componentes e Implantação) também não são considerados neste artigo, uma vez que a proposta é incentivar a elaboração dos testes desde o início do desenvolvimento e não apenas quando as características de implementação são acrescentadas aos artefatos.

Carniello et al.(2004) propõe um critério de teste que deriva casos de teste a partir da estrutura definida para os casos de uso, diferenciando-se de outros trabalhos que derivam casos de teste das especificações dos casos de uso (teste funcional). Já Bertolino et al.(2004) apresenta uma proposta para derivar casos de teste a partir dos Diagramas de Seqüência e Estados. Em Offutt and Abdurazik(1999) é apresentada uma ferramenta que gera casos de testes usando apenas o Diagrama de Estados, sendo que os autores apontam como trabalho futuro a utilização do Diagrama de Classes para expandir os testes definidos. Seguindo esta linha, em Aertryck and Jensen(2003) é apresentada uma ferramenta que considera os Diagramas de Classes e Estados na elaboração dos testes, através da identificação das classes de equivalência como critério de teste.

Por outro lado, existem trabalhos como Elaasar and Briand(2004) que descrevem procedimentos para garantir a consistência dos modelos, revelando problemas de projeto e uso incorreto da UML. Também existem trabalhos como Souza(2003) que descreve procedimentos para elaborar diagramas UML com características de testabilidade, ou seja, que apresentam informações relevantes para as atividades de teste. Para tanto, é prevista a necessidade de elaborar especificações complementares aos diagramas, de forma a expandir as contribuições fornecidas para os testes.

Diante deste cenário é incontestável a importância dos diagramas UML para as atividades de teste e, conseqüentemente, do tema abordado para obtenção de um sistema de qualidade. Porém, atividades como planejamento e execução dos testes também

devem ser consideradas, visando estabelecer uma estratégia adequada para sua efetivação de forma sistemática. Assim, o presente artigo complementa os trabalhos existentes uma vez que estabelece diretrizes para a realização de testes de sistemas de software OO, seguindo uma estratégia definida. Como resultado, as atividades relacionadas com o teste são feitas de forma estruturada e, conseqüentemente, têm sua qualidade melhorada.

3. Diretrizes para Teste de Sistemas OO

A estratégia de teste constitui um ponto importante para o processo de teste de software, pois estabelece a base para a elaboração do seu Plano de Teste. A estratégia apresentada, neste artigo, foi concebida com base nas experiências das autoras e seguindo uma série de propostas da literatura (Binder(2000), Colanzi(1999) e Pressman(2002)). Assim, é estruturada através dos seguintes tipos de testes: classe, integração, validação e sistema. Neste artigo, o teste de sistema não é abordado, pois sendo um teste de caixa preta, a sua condução independe da estruturação interna do software.

Para a realização de testes de classe e integração, o ambiente de teste será configurado utilizando os objetos *drivers* e *stubs*, conforme proposto em Binder(2000) e Pressman(2002). O objeto *driver* estimula o conjunto de objetos em teste, através do envio de mensagens com os parâmetros necessários. O objeto *stub* desempenha o papel da parte de software que é estimulada pelo conjunto de objetos em teste.

3.1. Teste de Classe

O objetivo deste tipo de teste é verificar o comportamento da classe para garantir o seu correto funcionamento. No Plano de Teste de Classes, a ordenação das classes para os testes deve considerar a prioridade estabelecida para cada caso de uso, uma vez que a integração das classes vai ser feita por caso de uso. Tal prioridade é definida no modelo de especificação proposto em Souza(2003), juntamente com informações como: descrição geral, atores envolvidos, pré e pós condições, dados de entrada, fluxos principal e alternativos, entre outras.

O conjunto de classes de cada caso de uso deve estar testado para que a integração possa ser realizada. Assim, as classes que participam dos casos de uso mais prioritários devem ser testadas antes das demais. Definida a seqüência de teste das classes, para cada uma delas devem ser realizados três tipos de atividade, de forma a garantir o seu bom funcionamento: análise das estruturas de herança, teste de método e teste de estados.

3.1.1. Análise das Estruturas de Herança

Em uma estrutura de herança é importante salientar que devem ser declarados, nas subclasses do Diagrama de Classes, apenas os métodos que serão redefinidos e os seus métodos exclusivos. Dessa forma, a partir desta estrutura, é possível identificar aqueles métodos que obrigatoriamente deverão ser testados no contexto da subclasse. Em Souza (2003) é proposto um modelo para especificação dos métodos, o qual apresenta uma breve descrição do seu papel, um pseudocódigo, seus dados de entrada e saída, bem como a existência de comunicação com outros métodos. Para os atributos também é proposto um modelo para sua especificação que define o tipo de informação a ser armazenada e o limite dos valores possíveis dentro do tipo estabelecido.

O contexto das subclasses deve ser analisado, para verificar como elas manipulam os atributos e os métodos herdados, auxiliando na identificação da necessidade dos testes de regressão dos métodos testados no contexto da superclasse, de definir novos casos de teste específicos para as subclasses e de definir casos de teste que coloquem à prova as características de polimorfismo.

As diretrizes para a análise das estruturas de herança são:

- Identificar as estruturas de herança, através do Diagrama de Classes;
- Analisar as funções dos métodos específicos das subclasses para identificar a necessidade ou não de realizar o teste de regressão na superclasse, além do teste de método no contexto da subclasse que é obrigatório. O teste de regressão é efetuado toda vez que um método da subclasse alterar algum atributo definido na superclasse;
- Analisar a possibilidade de reutilizar os casos de teste caixa preta definidos para a superclasse previamente testada, quando o método analisado na subclasse representar uma redefinição do método da superclasse. Salienta-se que, mesmo sendo possível reutilizar os casos de teste, o conjunto deve ser complementado de forma que a funcionalidade adicional seja adequadamente avaliada;
- Analisar se o método desejado é executado quando uma determinada mensagem é disparada, no caso de ser considerado um método polimórfico. Binder(2000) sugere que todos os tipos possíveis de objetos sejam testados pelo menos uma vez para cada método polimórfico.

Com a análise das estruturas de herança, é possível evitar retrabalho, através da identificação dos casos de teste, utilizados nas superclasses, que podem ser reutilizados nos métodos das subclasses.

3.1.2. Teste de Método

Após a análise das estruturas de herança, o teste de método deve ser realizado. Este teste avalia a consistência dos métodos das classes, viabilizando posteriormente os testes de estados e de integração.

As diretrizes definidas para o teste de método são:

- Selecionar uma classe do caso de uso considerado;
- Identificar os métodos da classe, através do Diagrama de Classes;
- Definir uma seqüência de execução para os métodos a serem testados;
- Definir um objeto *driver* para disparar o método desejado. Os dados de entrada necessários para a sua execução devem ser extraídos da especificação dos métodos;
- Definir objetos *stubs* para aqueles métodos que possuem comunicação interclasse, para simular a interação correta entre eles, além de retornar um valor, se necessário. Para determinar o valor de retorno, a especificação do método disparado deve ser consultada para obter o dado de saída que deve ser retornado após sua execução;
- Definir casos de teste com base em um critério de teste adotado (Souza(2003));
- Executar e acompanhar o teste de método com o auxílio do Diagrama de Atividades, analisando se as ações definidas no diagrama são efetivamente realizadas;

- Verificar se o método realizou a função esperada com base em sua especificação;
- Verificar se o resultado esperado com a execução dos casos de testes foi alcançado;
- Repetir os passos anteriores para todas as classes.

3.1.3. Teste de Estados

As classes que possuem um Diagrama de Estados associado devem ser identificadas para se definir o teste capaz de verificar a seqüência de estados.

As diretrizes definidas para o teste de estados são:

- Selecionar as classes com Diagrama de Estados associado para serem testadas;
- Definir os valores adequados para os atributos da classe testada, de forma que o estado inicial seja alcançado. Isso é feito analisando os limites dos valores aceitos para os atributos, os quais foram definidos na especificação dos atributos, e os estados representados no Diagrama de Estados;
- Definir um objeto *driver* para gerar a seqüência de disparo dos métodos, de forma que os diferentes estados possíveis da classe sejam alcançados. Para tanto, o objeto *driver* deverá solicitar a atribuição dos valores definidos na diretriz anterior. Em seguida, o objeto *driver* deve disparar a seqüência de métodos equivalente às ações representadas no Diagrama de Estados que causa as transições desejadas. Para tanto, deve-se analisar quais são os dados de entrada necessários para execução dos métodos, através de suas respectivas especificações;
- Definir objetos *stubs* para aqueles métodos que possuem comunicação interclasse, simulando a interação correta entre eles, além de retornar um valor quando necessário. Os *stubs* definidos para os testes de métodos podem ser reutilizados neste momento, caso o valor de retorno estabelecido seja adequado, visto que, para o teste de estados, pode existir valor específico a ser considerado para que o estado pretendido seja alcançado. Portanto, a especificação do método disparado deve ser consultada para obter o dado de saída a ser retornado após sua execução;
- Definir casos de teste com base em um critério de teste adotado (Colanzi(1999), Offutt and Abdurazik(1999));
- Executar o teste de classe e acompanhar a sua execução com o auxílio do Diagrama de Estados para verificar se as transições realizadas nos testes refletem o esperado. Para tanto, uma determinada transição deve ocorrer somente quando a condição estipulada for atendida e as ações disparadas devem levar ao estado desejado;
- Verificar se o resultado esperado com a execução dos casos de testes foi alcançado;
- Repetir os passos anteriores para todas as classes que possuem Diagrama de Estados.

3.2. Teste de Integração

O objetivo deste tipo de teste é garantir que a comunicação entre as classes é realizada corretamente. As classes são integradas por casos de uso, seguindo sua prioridade.

As diretrizes definidas para o teste de integração são:

- Selecionar um caso de uso, com base na prioridade estabelecida;

- Identificar o Diagrama de Seqüência relativo ao caso de uso a ser testado, para identificar as classes participantes. Caso o Diagrama de Seqüência contenha um *package* é necessário, primeiramente, identificar o Diagrama de Seqüência relativo ao *package* (Souza(2003)). Em seguida, verificar se a integração dos elementos que compõem o *package* já foi testada, o que possibilita testar o Diagrama de Seqüência relativo ao caso de uso desejado. Caso contrário, a integração das classes que compõem o *package* deve ser testada;
- Identificar os métodos que disparam as mensagens representadas no Diagrama de Seqüência, analisando o Diagrama de Classes;
- Identificar o Diagrama de Atividades relativo ao Diagrama de Seqüência;
- Definir um objeto *driver* para disparar o método que inicia a execução do caso de uso. Através da solicitação externa representada no Diagrama de Seqüência, é possível identificar os dados de entrada que o *driver* deve gerar;
- Analisar se a integração pode ou não ser feita sem o auxílio de *stubs*, conforme a complexidade das interações. Caso a utilização de *stubs* seja necessária, então:
    - Definir objetos *stubs* para as classes que não serão integradas no momento. À medida que as comunicações entre os objetos reais e os *stubs* tiverem sido testadas e as mudanças, quando necessárias, tiverem sido realizadas, os *stubs* devem ser substituídos pelos respectivos objetos reais. Cabe ressaltar que os objetos *stubs* definidos durante os testes de método e de estados das classes analisadas podem ser reutilizados, caso o valor de retorno ainda seja consistente;
- Definir casos de teste com base em um critério de teste adotado (Colanzi(1999)). Salienta-se que os casos de testes devem ser elaborados visando, também, as restrições de cardinalidade e das estruturas de agregação do Diagrama de Classes;
- Executar o teste de integração e acompanhar sua execução com o auxílio dos Diagramas de Seqüência e de Atividades. O Diagrama de Seqüência permite verificar se o método esperado de um determinado objeto foi realmente acionado e se a seqüência de mensagens trocadas é seguida. O Diagrama de Atividades possibilita o acompanhamento da realização das atividades esperadas, com o intuito de certificar que a seqüência de atividades estabelecida é seguida e as condições e os instantes determinados para a ocorrência de comunicação entre os objetos são respeitados;
- Verificar se o resultado esperado com a execução dos casos de testes foi alcançado;
- Repetir os passos anteriores para todos os casos de uso.

Após todas as interações entre os objetos serem testadas, as devidas alterações (quando existirem) serem implementadas e os testes de regressão necessários serem realizados, deve-se iniciar o teste de validação.

3.3. Teste de Validação

O objetivo deste tipo de teste é verificar se os requisitos funcionais foram atendidos.

As diretrizes definidas para o teste de validação são:

- Definir cenários (seqüência de casos de uso) para simular a execução do sistema;

- Selecionar um cenário;
- Selecionar um caso de uso do cenário;
- Utilizar as pré-condições da especificação do caso de uso selecionado, para definir os passos que levam o sistema ao estado inicial;
- Identificar os dados de entrada necessários para iniciar a execução do caso de uso, analisando a especificação de casos de uso;
- Definir os casos de teste, com base em um critério de teste adotado (Carniello et al. (2004), Chaim et al. (2003), Colanzi (1999));
- Verificar se o resultado esperado com a execução dos casos de testes foi alcançado. Este resultado é extraído da pós-condição descrita para o caso de uso;
- Repetir os passos para todos os casos de uso;
- Repetir os passos para todos os cenários.

4. Resultados Obtidos

A melhoria da qualidade de um sistema de software está fortemente ligada à execução sistemática de testes pois, desta forma, pode-se manter um controle sobre a verificação e validação das informações geradas. As diretrizes apresentadas neste artigo contribuem para a efetivação das diferentes atividades envolvidas durante os testes de sistemas de software, uma vez que direcionam a definição e a realização sistemática destas atividades. As diretrizes apresentadas foram aplicadas no desenvolvimento de um sistema acadêmico de pequeno porte, o qual auxilia o processo de matrícula dos alunos em disciplinas trimestrais e permite a análise da viabilidade ou não de oferecimento de determinada disciplina diante do número de alunos interessados. Observou-se que a adoção das diretrizes aumentou o tempo necessário para a elaboração dos diagramas, em razão das especificações textuais complementares. Por outro lado, foi possível identificar os seguintes pontos positivos:

- Padronização das atividades: o estabelecimento dos procedimentos a serem seguidos durante os testes contribui para que as atividades sejam previsíveis, possibilitando tomar decisões estratégicas em relação ao andamento do projeto;
- Confiança nos resultados obtidos: a indicação das contribuições dos diagramas UML para as atividades de teste contribui para a definição de testes corretos e adequados, uma vez que indica quais informações são relevantes para cada tipo de teste considerado. Assim, o uso das diretrizes propostas contribui para a garantia da qualidade dos testes realizados;
- Adequação das diretrizes: a descrição das atividades que compõem as diretrizes propostas possibilita a realização de um processo de melhoria contínua destas atividades, uma vez que alterações e/ou inclusões podem ser realizadas constantemente. Sendo assim, a documentação das diretrizes permite que suas atividades sejam avaliadas e aperfeiçoadas sempre que necessário.

Com isso, constatou-se que o conjunto de diretrizes apresentado contribuiu para a melhoria da qualidade do sistema desenvolvido, o que possibilitou maior confiança nas informações fornecidas por tal sistema.

5. Conclusão e Trabalhos Futuros

Considerando que um dos problemas enfrentados pelas empresas é a dificuldade de desenvolver e implantar sistemas de informação de qualidade, este artigo estabelece diretrizes para reduzir o esforço na definição dos testes e os custos relacionados, bem como sistematizar a sua realização, apontando de onde extrair as informações necessárias aos testes e a seqüência de realização das atividades, contribuindo para sua eficácia e efetivação.

Como trabalhos futuros, possíveis refinamentos nas diretrizes propostas podem ser considerados, buscando melhorar os resultados obtidos. Uma possibilidade é avaliar os critérios de teste existentes para sistemas OO, buscando expandir as diretrizes. Além disso, uma vez que as empresas necessitam construir sistemas de qualidade, sem comprometer os prazos estabelecidos inicialmente para o projeto, as características de diferentes tipos de sistemas estão sendo estudadas para identificar um conjunto mínimo de diagramas, com suas respectivas especificações textuais, que deve ser elaborado, sem comprometer a qualidade dos testes e reduzindo o tempo de desenvolvimento.

Referências

Aertryck, L.V. and Jensen, T. (2003) "UML-CASTING: Test Synthesis from UML Models using Constraint Resolution". In: Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'2003), Irisa, Rennes (France), 15 p.

Bertolino, A., Marchetti, E. and Muccini, H. (2004) "Introducing Reasonably Complete and Coherent Approach for Model-based Testing". In: International Workshop on Test and Analysis of Component Based Systems (TACoS'04),Barcelona,Spain, 12 p.

Binder, R.V. (2000), Testing Object-Oriented Systems, Addison-Wesley.

Booch, G., Rumbaugh, J. and Jacobson, I. (1999), The Unified Modeling Language - User Guide, Addison-Wesley.

Carniello, A., Jino, M. and Chaim, M.L. (2004) "Structural Testing with Use Cases". In: Workshop em Engenharia de Requisitos (WER04), Tandil, Argentina, p. 140-151.

Carr, N.G. (2003) "IT doesn't matter". In: Harvard Business Review, v.81, n.5, p.41-49.

Colanzi, T.E. (1999) "Uma Abordagem Integrada de Desenvolvimento e Teste de Software Baseada na UML", São Carlos. Dissertação - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.

Elaasar, M. and Briand, L. (2004) "An Overview of UML Consistency Management". In: Technical Report – Department of Systems and Computer Engineering, Ottawa, Canada, SCE-04-18.

Offutt, J. and Abdurazik, A. (1999) "Generating Tests from UML Specifications". In: 2nd International Conference on Unified Modeling Language: beyond the standard (UML'99), Fort Collins, CO, USA, p. 416-429.

Pressman, R.S. (2002), Engenharia de Software, McGraw-Hill, 5.ed.

Souza, R.C.G. (2003) "Características de Testabilidade nos Diagramas UML (Unified Modeling Language): Apoio aos Testes de Sistemas de Software Orientados a Objetos", São Paulo. Tese – Escola Politécnica, Universidade de São Paulo.