1
Introdução

Testar é o conjunto de tarefas ou passos executados para verificar se um produto ou serviço atende à sua proposta. Dessa forma, a execução de testes em um programa contribui para a melhoria da qualidade do código. Testes são muito utilizados em processo de desenvolvimento ágeis (Agile Manifesto, 2001). Algumas das características dos processos de desenvolvimento de software que seguem o paradigma de metodologias ágeis, como o "*extreme programming*" (Beck e Andres, 2004), são a entrega frequente de novas funcionalidades, ciclos de desenvolvimento curtos e rapidez de resposta a mudanças nos requisitos ao longo do projeto. Uma das consequências do desenvolvimento iterativo com ciclos curtos e da tolerância a mudanças é a necessidade de testes frequentes, em particular, torna-se necessária uma elevada frequência de execução de testes de regressão, a fim de determinar se novas funcionalidades não comprometeram o correto comportamento de outros requisitos anteriormente implementados e já validados. Caso os testes não sejam automatizados, os custos desta abordagem de desenvolvimento tornam-se proibitivos e propensos a falhas humanas.

Uma técnica bastante utilizada nesses processos é o desenvolvimento dirigido por testes (DDT, ou no original *test driven development* - TDD). Nesta técnica os testes automatizados de cada artefato devem ser planejados e escritos antes da sua codificação. A ideia por trás dessa prática é fazer com que o desenvolvedor, ao verificar quais os testes necessários, reflita sobre o problema a ser resolvido e como resolvê-lo. Segundo os proponentes isso implicaria em um código de melhor qualidade. Outro benefício seria a identificação de defeitos nas primeiras etapas do desenvolvimento, onde o custo para removê-los é menor, pois seus efeitos colaterais estarão espalhados por menos linhas de código ou módulos do sistema (Boehm e Basili, 2001).

Um dos efeitos colaterais do DDT é que o módulo (classe) ou script de teste passa a ser uma especificação através de exemplos, em que os exemplos são efetivamente verificáveis por meios automáticos (Ostroff, 2004). É claro que

especificação por meio de exemplos não é garantia de corretude, mas a experiência mostra que tende a ser melhor do que o estado da prática hoje encontrado. Um dos problemas observados é o fato de DDT ser fortemente voltado para teste de unidades (módulos). Entretanto, o comportamento que interessa a um usuário é estabelecido por uma composição envolvendo diversos módulos. Uma das propostas para resolver este problema é partir para o desenvolvimento dirigido por comportamento.

O desenvolvimento dirigido por comportamento – DDC (no original *behavior-driven development* – BDD) (North, 2006) surgiu como uma extensão ao DDT, tendo por meta funcionar como uma técnica para a identificação de funcionalidades do sistema em desenvolvimento, levando a uma especificação de requisitos que possa ser lida tanto por clientes quanto por desenvolvedores. Entretanto, o mais interessante é que as especificações DDC podem ser mapeadas para testes. Existem diversas ferramentas, algumas descritas em seção mais adiante, que realizam esta conversão de acordo com regras específicas de cada ferramenta. Ao serem transformadas em testes, teríamos especificações executáveis que validam as funcionalidades do sistema.

DDC procura fazer uso de uma linguagem comum (North, 2006) entre os envolvidos no projeto para a descrição dos comportamentos e funcionalidades do sistema, a fim de facilitar a comunicação e o entendimento dos requisitos e dos cenários, no qual o objetivo é especificar os critérios de aceitação de um aspecto de uma dada funcionalidade. A linguagem utilizada por DDC é uma linguagem natural semi-estruturada (no nosso caso português). Entretanto, a sub-linguagem proposta por North é bastante abstrata, podendo deixar de identificar uma série de aspectos importantes para a implementação, tais como tratamento de situações anômalas e requisitos não-funcionais. Também pode gerar, para cada funcionalidade, um grande número de cenários interdependentes sem que esta interdependência esteja explícita, dificultando a visão do problema como um todo, bem como a avaliação da cobertura dos testes. A redação de todos os cenários possíveis também pode ser muito trabalhosa e propensa a erros humanos devido à natureza fatigante da tarefa.

Surgiu, então, a ideia de combinar elementos presentes em DDC com especificações baseadas em casos de uso (Jacobson, 1992) e tabelas de decisão. A razão para tal é que casos de uso fazem parte da UML (UML, 1997) que

atualmente é uma linguagem bastante difundida no meio empresarial, além de serem capazes de especificar os requisitos funcionais de um sistema. Entretanto, a especificação baseada em casos de uso tende, em geral, a ser excessivamente informal para poder ser utilizada como base para o DDC.

Neste trabalho o interesse principal está em pesquisar como casos de uso e tabelas de decisão podem ser integrados para a geração semi-automática de massas de testes funcionais automatizados e o quanto essa abordagem pode reduzir o esforço necessário para a geração da massa de testes. Os casos de uso devem estender o desenvolvimento dirigido por comportamento no que diz respeito ao uso de uma linguagem natural (português) restrita e semi-estruturada. O objetivo é viabilizar a leitura e compreensão de casos de uso por clientes ou pessoas com pouco ou nenhum treinamento em computação e, ao mesmo tempo, permitir que estes sejam especificações para os testes. Estes casos de uso deverão poder ser editados por um técnico e, em seguida, utilizados para gerar uma tabela de decisão a partir da sua interpretação manual. A tabela de decisão gerada será utilizada para produzir automaticamente, através de ferramenta desenvolvida, arquivos de testes "caixa preta" que possam servir como testes funcionais e de regressão para sistemas web. Idealmente estes testes seriam automatizados. Os comandos dos testes envolvendo interação com elementos de interface podem ser automatizados através de adequadas ferramentas de "*capture and replay*" (Araújo e Staa, 2009).

1.1. Objetivos

O objetivo da dissertação é propor um procedimento e ferramentas para a geração semi-automática de testes funcionais para sistemas web a partir da integração de casos de uso, tabelas de decisão e *frameworks* de automação de testes, com vistas à produção de especificações mais rigorosas. Este procedimento é apoiado por ferramentas desenvolvidas com base no trabalho de (Lachtermacher, 2010).

Um caso de uso definiria o comportamento de uma funcionalidade ou característica do sistema na sua trilha principal e em seus caminhos secundários (erros e afins). Nesse contexto, seria interessante desenvolver uma forma de escrever casos de uso que utilizassem uma linguagem semi-estruturada (Staa,

2010a) (Díaz et al, 2004) (Heumann, 2001) com um vocabulário restrito ao domínio da aplicação, assim como em DDC. Essa linguagem deve ser um subconjunto do português com um viés formal, sem, entretanto, dificultar a leitura e compreensão por pessoas leigas em computação. Idealmente esta linguagem deveria viabilizar a escrita por pessoas com pouco ou nenhum treinamento em computação.

A descrição do caso de uso deve seguir um formato estruturado de tal forma que possa ser possível, a partir do caso de uso, identificar e extrair os elementos necessários para a criação de uma tabela de decisão. Em especial, como estamos interessados em simular a interação de um usuário do sistema com a interface, a linguagem deve permitir identificar que elementos de interface são exercitados em cada teste.

A geração da tabela de decisão se dá de forma manual por um técnico, apoiado por uma ferramenta para edição da tabela, através da análise dos passos descritos no caso de uso. A tabela de decisão gerada seria utilizada para a criação automática dos casos de teste. Estes testes devem ser automatizados e voltados para interfaces de sistemas web.

Os resultados esperados são:

- Pesquisar uma linguagem natural restrita (português estruturado) possuindo um viés formal para a redação de casos de uso que se adeque ao paradigma de DDC. A descrição dos fluxos dos casos de uso deve permitir a identificação dos elementos com os quais o usuário interage, a fim de possibilitar a criação da tabela de decisão e dos testes.
- Desenvolver um procedimento manual para criação de tabelas de decisão a partir de um formulário de caso de uso.
- Desenvolver procedimentos automáticos de conversão de tabelas de decisão para testes funcionais automatizados na linguagem de programação Java. Estes testes seriam voltados para interfaces de sistemas web e seriam automatizados por JUnit (JUnit, 2009). A interação com a interface seria automatizada através da biblioteca Selenium (Selenium, 2009) para Java.
- Avaliar a proposta levando em conta um sistema real.

Por procedimentos entendemos ações de pessoas (desenvolvedores e/ou usuários) apoiadas por protótipos de ferramentas capazes de servir como prova de conceito.

Não faz parte deste estudo a avaliação da linguagem de especificação quanto a sua adequação a leitores e redatores leigos em computação.

O restante deste documento está estruturado da seguinte forma: na seção 2 é apresentado o estado da arte para DDC, para a técnica "*capture and replay*", e para casos de uso e tabelas de decisão com vistas à produção de testes; a seção 3 apresenta o processo proposto para a geração dos testes; na seção 4 são explicadas as ferramentas desenvolvidas; a seção 5 apresenta a aplicação do processo em um sistema real e seus resultados e comparação com outras técnicas; na seção 6 é feita uma conclusão sobre o trabalho realizado; a seção 7 apresenta uma lista de trabalhos futuros e, finalmente, na seção 8 é listada a bibliografia.