1. Introdução

A fase de testes é uma das mais importantes no ciclo de desenvolvimento de um software, pois visa expor a presença de defeitos e aumenta a confiança e a justificativa de sua corretude (MILER et al., 1992). Entretanto, o processo de testes é trabalhoso e caro, consumindo entre 35 e 50% de seu custo total de desenvolvimento (BENDER, 1996), quando realizado da forma tradicional^{1,2}. E a confecção manual de testes, por se mostrar muitas vezes uma atividade tediosa e ineficiente (BINDER, 2000) – geralmente apoiada em práticas *ad hoc* e dependente da habilidade e atenção de seus criadores –, contribui para o aumento destes custos. Assim, sua geração de forma automatizada torna-se extremamente valiosa e até mesmo necessária.

Diversos trabalhos, como os analisados no Capítulo 3, têm realizado esta automação usando diferentes abordagens, tentando contribuir para a redução dos custos de desenvolvimento e a melhoria da qualidade do software construído. Poucos, porém, se concentram na verificação da adequação do software em relação aos seus requisitos, que é uma das principais formas de atestar sua qualidade (CROSBY, 1979).³ Fazendo uso dos requisitos, pode-se diminuir a explosão combinatória de possíveis caminhos percorráveis no uso do software, diminuindo o tempo e os recursos necessários para a geração dos testes, além de se concentrar nos casos importantes para seu maior interessado, o usuário.

Uma vez que a descrição textual de casos de uso é uma forma bem conhecida de documentar os requisitos (funcionais e de domínio) de um software, seu uso torna-se interessante como ponto de partida para a geração de testes funcionais. Por conta disto, alguns trabalhos, como os de Gutiérrez *et al.* (2008), Caldeira (2010) e Pessoa (2011), têm utilizado casos de uso para a geração destes testes, construindo ferramentas que extraem suas informações empregando processos

¹ (BEIZER, 1990) e (NIST, 2002) chegam a estimar um custo de até, respectivamente, 50% e 80% para o desenvolvimento da forma tradicional.

² Segundo (MAGALHÃES, 2009), quando o desenvolvimento é baseado em técnicas formais, empregando-se, por exemplo, o uso extensivo de assertivas e de verificadores estruturais, o custo estimado fica entre 15% e 25%.

³ Supondo, é claro, que os requisitos estejam corretos e de acordo com necessidade do usuário.

semiautomatizados ou totalmente automatizados. Muitos destes processos, todavia, têm deixado aspectos importantes de lado, que podem ser significativos para a aplicação prática das soluções criadas em aplicações reais.

A presente dissertação identifica as principais deficiências de trabalhos correlatos e apresenta novos processos e algoritmos para suprir algumas delas, através de uma solução em forma de ferramenta.

1.1. Objetivos

O presente trabalho tem como objetivo a construção de uma solução que:

- Documente, usando casos de uso, os requisitos funcionais e de domínio de um software;
- Permita o detalhamento das regras de negócio dos casos de uso, sobretudo as que possam servir como meio de verificar o software de forma automática;
- Possibilite o uso de fontes de dados externas, como bancos de dados, na definição das regras de negócio, para que seja possível o uso de dados com estrutura idêntica aos reais (de produção) nos testes;
- Efetue a geração automática de cenários;
- Efetue a geração automática de casos de teste semânticos;
- Efetue a geração automática dos valores usados nos testes;
- Efetue a geração automática de oráculos;
- Forneça um alto grau de cobertura dos requisitos funcionais nos testes gerados, possibilitando realizar testes de elevada eficácia (onde eficácia é o percentual de defeitos encontrados considerando o total de defeitos existentes);
- Possibilite a exportação dos casos de teste semânticos valorados e com oráculos (CTSVO) para formatos independentes de linguagens de programação e de *frameworks* de testes;
- Realize a conversão dos CTSVO para código-fonte de módulos de teste e execute este código;

- Colete os resultados da execução, analise-os e verifique o atendimento do software sob teste (SST) aos requisitos especificados.

No Capítulo 4 é realizada uma especificação detalhada sobre os assuntos envolvidos na construção desta solução.

1.2.Resultados esperados

Os principais resultados esperados para o emprego da ferramenta são:

- **Aumento do percentual de defeitos encontrados (eficácia) no conjunto de defeitos existentes**, comparado à criação de testes funcionais com uso de ferramentas de *capture-and-replay* ou que geram testes a partir de requisitos descritos como casos de uso, uma vez que essas não geram dados de teste nem oráculos (que exploram as regras de negócio do software);
- **Diminuição do número de fluxos alternativos** a serem criados, visto que fluxos alternativos que descrevem exceções por entradas incorretas serão cobertos pelos casos de teste gerados com base nas regras de negócio. Este assunto é detalhado na seção 4.3;
- **Redução do esforço necessário para geração dos testes funcionais**, comparado ao uso de *capture-and-replay*;
- **Redução significativa do esforço humano** necessário para criar e executar suítes de teste;
- **Possibilidade de atestar a conformidade do software sob teste em relação aos seus requisitos** (funcionais e de domínio);
- **Contribuição para a área de *Test-Driven Development***, considerando que a ferramenta fornece um meio de gerar dados de teste, oráculos e testes funcionais de forma totalmente automática, verificar a correspondência do software sob teste com a especificação e realizar Desenvolvimento Dirigido por Testes de Aceitação (DDTA). No DDTA, o software vai sendo especificado, construído e testado incrementalmente. Para construir uma funcionalidade, ela primeiro é especificada, depois tem seus testes funcionais criados e então é construída para passar nos

testes. Com a presente ferramenta, esse processo pode ocorrer de forma mais simples e mais rápida.

Estes resultados culminam na **redução de trabalho** realizado pela equipe de desenvolvimento para realização de testes funcionais e na **melhoria da "qualidade externa"** (percebida pelo usuário final) do software.

1.3.Organização dos capítulos seguintes

Os próximos capítulos estão organizados da seguinte forma: o **Capítulo 2** descreve brevemente a terminologia de assuntos discutidos nos capítulos seguintes. O **Capítulo 3** apresenta o estado da arte, relacionado ao assunto da presente dissertação. O **Capítulo 4** detalha todo o processo realizado pela solução construída, para atingir seus objetivos, sem se ater às particularidades de projeto e construção desta. O **Capítulo 5** aborda os aspectos de projeto e desenvolvimento da ferramenta. O **Capítulo 6** apresenta uma demonstração da viabilidade técnica da ferramenta construída. O **Capítulo 7** apresenta as conclusões acerca do presente trabalho. Por fim, o **Capítulo 8** apresenta nossos projetos futuros.