8. Projetos Futuros

Este capítulo apresenta ideias e técnicas que podem ser aplicadas para melhorar a solução construída.

8.1.Melhorias

Inicialmente, desejamos criar meios de superar as atuais *limitações da ferramenta* (descritas na seção 3.3.3), sobretudo aquelas relacionadas aos passos do fluxo e às regras de negócio. Ao adicionar o suporte ao uso de condicionais nos passos de um fluxo, acreditamos que a solução poderá representar mais situações de uso do SST, em que a passagem por um determinado caminho de execução será realizado somente se certa condição for satisfeita. Da mesma forma, o uso de estruturas de repetição simples, como um laço que executa um número determinado de vezes, tornaria possível testar a passagem por caminhos que funcionam de forma diferente dependendo da quantidade de vezes em que foi percorrida (como, por exemplo, um caso de uso de *login* que pode terminar a execução do SST após fornecer credenciais inválidas por três vezes). Já a representação de expressões que envolvam cálculos, pode estender o potencial das regras de negócio, possibilitando a descrição de operações mais complexas. Estas expressões poderiam ser escritas com uso de uma linguagem que pudesse ser interpretada pela ferramenta no momento da geração de testes (idealmente Lua, ou JavaScript).

Adicionalmente, poderíamos melhorar o processo de geração ou combinação de cenários, visando gerar uma quantidade menor de testes, diminuir o tempo total de sua execução e se concentrar nos cenários mais importantes. As seções 8.1.1 e 8.1.2 discutem estas melhorias.

8.1.1.Geração de cenários para cada caso de uso

Como discutido na seção 4.4, a geração de cenários combina todos os fluxos do caso de uso pelo menos uma vez, garantindo um bom nível de cobertura para a

geração de testes. Isto é um atributo fortemente desejável para verificar o SST antes da liberação de uma versão para o usuário final. Durante seu desenvolvimento, entretanto, pode ser interessante diminuir a cobertura realizada, para que o processo de execução de testes ocorra em menor tempo.

Para isto, propõe-se o emprego de duas técnicas:

1. **Uso do valor da importância** do fluxo (seção 4.2.2), de forma que somente os fluxos com importância maior ou igual a um valor definido pelo analista/testador serão selecionados para criação de cenários;
2. **Indicação da não-influência** de estados de determinados fluxos em outros, de forma que sua combinação não será necessária. Para realizar esta declaração, o analista/testador deve estar ciente dos falsos negativos, isto é, do risco de declarar uma não influência quando, na verdade, ela existe. Logo, esta técnica deve ser empregada com cuidado.

8.1.2.Combinação de cenários entre casos de uso

A combinação entre cenários de diferentes casos de uso, discutida na seção 4.5, também poderia empregar o **uso da importância** (vide seção anterior) para a seleção de cenários, minimizando o conjunto selecionado para possibilitar a execução rápida de testes.

Outra interessante (e mais importante) melhoria seria realizar a escolha dos cenários de forma **aleatória e baseada em histórico**. Nela, o combinador de cenários selecionaria apenas um cenário para combinação (pseudoaleatoriamente) e este cenário seria diferente a cada vez em que a geração dos testes fosse realizada (garantida pela consulta ao histórico de escolhas passadas). Desta forma, a cobertura da combinação de todos os cenários seria atingida *gradualmente*, chegando à cobertura completa ao longo do tempo.

8.1.3.Paralelização

Além das técnicas anteriormente discutidas, pode ser interessante modificar os algoritmos construídos para permitir sua **execução em paralelo**. Desta forma,

diminuiríamos seu tempo de execução, aproveitando o conjunto de processadores disponíveis.

As etapas de *geração de cenários para cada caso de uso*, de *transformação em código-fonte* e de *coleta, análise e apresentação dos resultados*, poderiam ser facilmente paralelizadas, uma vez que não possuem interdependências.

8.2.Avaliações

Podemos realizar uma vasta quantidade de avaliações, visando investigar o comportamento da ferramenta construída em seu uso prático, como, por exemplo:

- Esforço necessário para gerar testes, em relação a *capture-and-replay*;
- Esforço necessário para gerar testes, em relação a outras ferramentas de *model-based testing*;
- Eficácia dos testes gerados, em relação a outras ferramentas de *model-based testing*;
- Custo de manutenção dos testes gerados, em relação a outras ferramentas de *model-based testing*;
- Uso da ferramenta por terceiros (por exemplo, se um usuário conseguiu descrever todos os casos de uso e regras de negócio de seu software com a ferramenta, quais limitações foram encontradas, o que pode ser melhorado, etc.);
- Comparativo dos testes gerados, analisando quais tipos de teste encontram mais defeitos;
- Comparativo das formas de selecionar cenários e regras de negócio para a geração de testes rápidos, visando determinar qual delas possui o melhor custo/benefício.

8.3.Funcionalidades complementares

Algumas funcionalidades complementares podem aumentar a utilidade da ferramenta para o usuário, como:

- **Criar outras extensões da ferramenta**, para gerar código de teste para

outras linguagens e *frameworks*, aumentando sua abrangência;

- **Criar outros tipos de teste**, visando explorar outras condições de execução do SST e aumentar a eficácia dos testes gerados;
- **Permitir um testador criar casos de teste adicionais** (aos gerados automaticamente) com apoio da ferramenta, possibilitando explorar possíveis condições não previstas pelos testes gerados automaticamente;
- **Permitir um testador alterar os casos de teste semânticos**, através da ferramenta, possibilitando realizar a *manutenção dos testes* de forma independente de linguagem;
- **Gerar automaticamente protótipos de interface com o usuário**, a partir da descrição textual de casos de uso, permitindo dar rápido *feedback* ao usuário sobre o funcionamento do software. Esta funcionalidade foi construída numa versão passada da ferramenta, como um trabalho para a disciplina de Interfaces Inteligentes, da PUC-Rio, onde foi gerada uma interface *web* usando XHTML, CSS e JavaScript. Ela poderia ser reintroduzida e melhorada, permitindo criar interfaces para outras linguagens e *frameworks*;
- Permitir definir regras de negócio baseadas em consultas para arquivos nos formatos como CSV (*Comma Separated Value*), XML ou JSON, aumentando a abrangência das regras de negócio quanto às fontes de dados de teste.