7
Discussão dos Resultados

Neste trabalho podemos dividir os resultados gerados em três grupos: a ferramenta de testes, a linguagem criada e sua aplicação em um projeto real. Embora sejam bastante diferentes entre si, todo o trabalho foi realizado com o intuito de se estudar a possibilidade de utilização de linguagens de especificação de testes como linguagem de especificação dos requisitos do que se deseja construir. A seguir são apresentados os resultados obtidos para cada um dos diferentes grupos de resultados.

7.1.
A Ferramenta de Testes

O principal objetivo relacionado à criação da ferramenta de testes objeto desse trabalho era possuir um *framework* de testes que permitisse a sua extensão, de maneira suficientemente simples, para a modificação da linguagem de especificação dos testes, da forma como os resultados são apresentados e para realizar a sua integração com outras ferramentas. A experiência desse trabalho mostrou que a abordagem utilizada, onde os diversos módulos são ligados em tempo de execução através de técnicas de injeção de dependências, foi bastante eficiente no cumprimento desse objetivo. A interface definida foi simples o suficiente para não tornar a criação dos novos componentes demasiadamente complexa.

Todas as alterações necessárias para a criação da linguagem proposta no trabalho foram realizadas utilizando o módulo de linguagem padrão, sem a necessidade de se desenvolver um módulo totalmente novo, o que tornou essa tarefa bastante simples. Aparentemente, novos módulos de linguagem só precisariam ser criados caso se saia do modelo textual, embora mesmo que se crie uma linguagem gráfica, por exemplo, existe a possibilidade de criar módulos conversores que levem da linguagem gráfica para a linguagem textual configurada. O mesmo pode ser dito dos módulos de apresentação de resultados.

Entretanto, a maior ameaça à validade externa dessas afirmações reside no fato de todas as instanciações do *framework* realizadas terem contado com a participação da equipe que de fato desenvolveu a ferramenta. Seria necessário o envolvimento de outros desenvolvedores, com diferentes formações, na utilização da ferramenta para que fosse possível verificar de forma mais imparcial a simplicidade e clareza de seus pontos de extensão.

7.2. A Linguagem Baseada em Máquinas de Estado

Outro produto importante desse trabalho foi a criação de uma linguagem de especificação de testes capaz de ser utilizada também como linguagem de especificação do sistema que se deseja construir. Foi adotada uma linguagem puramente textual, na qual foram mesclados os comandos de definição de transições, imposição de estado e geração dos eventos com os comandos tradicionais da ferramenta de testes. Essa nova linguagem, por ser baseada em máquinas de estado, é mais próxima do domínio da aplicação, e já era usualmente empregada nas especificações dos comportamentos dos componentes reativos. A utilização de uma linguagem mais próxima do domínio e o fato dessa linguagem ser focada na especificação de comportamentos [4,8,9] evidenciam as características de orientação a comportamentos do trabalho. Entretanto, apenas a parte da linguagem responsável pela definição da máquina de estados foi capaz de manter o foco nos comportamentos. A parte de criação dos casos de teste, responsável por exercitar a máquina e buscar por falhas, foi demasiadamente semelhante às linguagens convencionais. Entretanto, como a intenção é que apenas a máquina de estados sirva como especificação dos comportamentos, acredita-se que os comportamentos tenham sido corretamente evidenciados. Essa abordagem, em conjunto com a escolha de nomes mais significativos para os comandos de teste tradicionais seria suficiente para, além de especificar o sistema de forma satisfatória, fornecer aos desenvolvedores os recursos necessários para a criação dos casos de teste.

Do ponto de vista da criação dos casos de teste, a linguagem foi bastante completa e nos momentos que alguma verificação além dos estados modelados se fez necessária, os outros comandos foram utilizados. A combinação de comandos de mais alto nível de abstração implementados por métodos no módulo específico

de testes com a possibilidade de criar seqüências de comandos que exercitem o comportamento que se deseja testar faz com que essa abordagem seja bastante natural para profissionais com alguma experiência no desenvolvimento de *software*.

Como linguagem de especificação, entretanto, o formato textual não foi eficaz. Tal formato representou grandes dificuldades já na etapa de modelagem das máquinas de estado. As linguagens gráficas tornam o processo de modelagem muito mais intuitivo, portanto foi necessário utilizar outra ferramenta com tais características nesse processo. As máquinas de estado em versão gráfica se mostraram bastante eficientes para expressar os comportamentos dos módulos testados para todos os integrantes do projeto.

O ideal para a linguagem, portanto, provavelmente seria uma abordagem híbrida, na qual a máquina de estados fosse modelada através de uma ferramenta gráfica, mas houvessem mecanismos para a criação dos casos de teste de forma textual, da mesma maneira como foram feitos nesse trabalho.

7.3. A Aplicação ao Projeto

A aplicação em um projeto real serviu para verificar as crenças que motivaram a criação da linguagem de especificação de testes e da ferramenta construída para utilizar a linguagem. A partir das experiências obtidas através da utilização, também foi possível perceber algumas das vantagens e desvantagens da aplicação das técnicas de desenvolvimento dirigido por comportamentos em uma situação real.

As primeiras impressões obtidas foram com relação às alterações necessárias no processo de desenvolvimento utilizado no projeto, já institucionalizado. A aplicação da ferramenta exigiu que a etapa de modelagem dos requisitos fosse realizada de forma mais cuidadosa, o que demandou um maior esforço da equipe nessa tarefa. Da mesma forma, os processos de criação dos *scripts* e dos módulos específicos de testes também contribuíram para que houvesse resistência à implantação dos novos processos. Vale lembrar que antes das modificações no processo nenhum tipo de teste automatizado era realizado. A aplicação de outras formas de testes automatizados, que também envolvessem a criação das especificações de testes antes da etapa de codificação, provavelmente

traria os mesmos custos para a equipe, de forma que a diferença de esforço entre a utilização de técnicas mais convencionais e a utilização da abordagem proposta nesse trabalho não pode ser corretamente avaliada.

A necessidade de cuidados com a co-evolução dos artefatos de modelagem das máquinas de estados e dos scripts de testes também dificultou bastante a adoção das novas técnicas, mesmo com o caso estudado nesse trabalho, que é relativamente pequeno. Enganos também ocorreram por conta dos comentários que os desenvolvedores deveriam fazer ao adicionar os artefatos contendo os modelos das máquinas de estados ao controle de versão. Estes comentários deveriam conter o código da tarefa relacionada, e seriam utilizados pelo sistema de acompanhamento de tarefas para vincular uma tarefa com os artefatos correspondentes. Comentários sem o código da tarefa ou com códigos trocados causaram algumas dificuldades para identificar os arquivos de modelagem correspondentes. Entretanto, pelo pequeno número de arquivos de modelagem, essa questão não foi muito impactante no projeto. Entretanto, talvez com um maior número de documentos de modelagem esse problema fosse sentido de forma mais negativa pela equipe.

Outra grande mudança no processo foi relativa ao procedimento de testes. Anteriormente os testes eram realizados de forma manual, seguindo roteiros de testes. Com a utilização da ferramenta, os testes passaram a ser realizados de forma automática, o que possibilitou a realização de testes mais freqüentes e com um menor esforço. A impressão com a utilização da ferramenta foi de que o esforço necessário para a criação dos modelos, *scripts* e módulos de teste foi facilmente superado pelas facilidades obtidas durante os testes. Neste sistema específico, os testes manuais eram especialmente complexos de serem realizados, pois envolviam o envio de coordenadas para o sistema através de uma ferramenta criada para esse fim. As coordenadas eram enviadas seguindo cenários onde se tentava reproduzir a situação que se desejava testar. Esse processo era realizado de forma seqüencial e falhas do testador ao seguir a seqüência tornavam necessário que se reiniciasse o procedimento, o que tornava a tarefa maçante e ainda mais propensa a erros. Acredita-se que somente com o esforço reduzido para a realização dos testes se tenha resultados que por si só justifiquem a adoção da ferramenta. Entretanto, como nenhuma outra ferramenta de automatização de testes era utilizada anteriormente, é bastante provável que outras ferramentas

apresentassem resultados semelhantes, de forma que não é possível afirmar sobre a utilidade da ferramenta sem compará-la de forma quantitativa com outras ferramentas.

No projeto, os scripts de testes não foram utilizados como especificação de forma isolada. Foram necessários também os artefatos de modelagem das máquinas de estado, uma vez que a linguagem de modelagem de máquinas de estado gráfica foi muito mais clara como especificação para a equipe de desenvolvimento e para a geração de documentos legíveis para os membros não técnicos.

Uma nova ferramenta foi necessária para a modelagem das máquinas de estado gráficas. A ferramenta deveria permitir que as máquinas fossem modeladas utilizando os diagramas de máquinas de estado propostos pela linguagem de modelagem UML 2 [15]. Existem diversas ferramentas disponíveis no mercado para tal função, e a escolhida foi a ferramenta JUDE Community versão 5.3 [35], por ser livre, de fácil utilização, e pela familiaridade de alguns dos integrantes do projeto com a ferramenta. A necessidade de uma nova ferramenta para essa função não foi uma barreira para a aplicação do novo processo.

Ao todo foram observadas vinte e três falhas, que foram causadas por vinte e um defeitos. Apenas duas dessas falhas foram encontradas após o componente ter sido considerado pronto, e foram mapeadas para dois defeitos. A quantidade de defeitos encontrados após a entrega, se comparados com os defeitos encontrados durante o processo de aceitação interno, leva a crer que a aplicação das técnicas foi eficiente para a eliminação precoce dos defeitos. Entretanto, como nenhuma técnica de automatização de testes era utilizada anteriormente, outras formas de testes automatizados provavelmente também obteriam resultados positivos sob esse aspecto. Seria necessário também possuir informações de medições da quantidade de falhas detectadas antes e depois da aceitação dos módulos anteriores à utilização da ferramenta e da linguagem, para que fosse possível realizar a comparação dos resultados obtidos antes e após a sua utilização no projeto.

Foi notado pela equipe que também houve uma maior facilidade para a correção dos defeitos causadores das falhas encontradas. Acredita-se que essa maior facilidade se deva ao fato das falhas terem sido encontradas, na maioria dos

casos, com a ferramenta de testes. Nesse aspecto, a ferramenta de testes teve como principais vantagens as seguintes características:

- permitir a recriação de forma simples das falhas encontradas, bastando para isso re-executar o *script* correspondente;
- fornecer uma forma simples para que o desenvolvedor responsável pelo diagnóstico do defeito analise a seqüência de passos que levaram a ocorrência da falha através da seqüência de comandos no caso de teste onde ela ocorreu;
- oferecer informações de diagnóstico, como a pilha de execução no momento da falha ou o estado atual da máquina após uma transição errada, que podem ser usadas para facilitar a detecção do defeito que ocasionou a falha percebida.

Portanto, quanto maior for o número de falhas encontradas nesse ambiente controlado, provavelmente menor será o custo para corrigi-las, uma vez que o diagnóstico é bastante facilitado e os defeitos são corrigidos de forma bastante prematura. Falhas apontadas após a entrega do software, a partir da fase de testes de aceitação do cliente, tendem a ter correções bastante custosas, uma vez que são necessárias informações dos usuários do sistema, que muitas vezes as fornecem de forma incompleta ou equivocada.

Existe também uma dificuldade relacionada ao ambiente de execução. Diferenças nas versões de softwares dos quais o sistema depende para funcionar, como servidor de banco de dados e o servidor de aplicações J2EE, podem causar problemas ainda desconhecidos. Por esse motivo, existem versões definidas para esses servidores. Entretanto, algumas vezes tempo é perdido por conta de falhas atribuídas ao sistema por equívocos nas versões utilizadas. É interessante também possuir os mesmos dados utilizados no ambiente em que ocorreu a falha, uma vez que em determinadas ocasiões o defeito pode só ser exercitado com um estado específico do banco de dados. Esse procedimento demanda recursos e tempo, uma vez que é necessário obter a base de dados junto ao cliente e importá-la no ambiente de desenvolvimento, para só então iniciar as tentativas de reprodução da falha. Nas falhas encontradas através da utilização da ferramenta esse processo possui esforço zero por construção, pois a falha foi observada já no ambiente de desenvolvimento, com uma base de testes própria e que pode ser utilizada para o diagnóstico do defeito.