Projeto 6.49

Suíte de testes de *interface* gráfica automatizados para testes de regressão em *builds* diários de um sistema

Daniel de Freitas Ferreira, Viviane do Nascimento Bernardo

dfreitas@fpf.br, vbernardo@fpf.br

Introdução

A atividade de testes em *software* foi considerada secundária por muito tempo. Geralmente era vista como um castigo para o programador ou como uma atividade onde não se deveria gastar muito tempo e investimentos. Contudo, essa visão foi mudada e os testes nos dias de hoje são executados desde o início do desenvolvimento de um *software* (MACORATTI, 2004).

O teste enfatiza a deficiência, por isso é muito mais interessante achar defeitos no início do desenvolvimento ao invés de no final, pois o custo para corrigir o problema é bem menor.

A finalidade dos testes para a detecção de defeitos é expor defeitos latentes em um *software* antes que ele seja entregue e, também, destina-se a validar se o *software* está seguindo a sua especificação (SOMMERVILLE, 2003).

Com o objetivo de incentivar e melhorar a atividade de testes foi criada uma suíte de testes de GUI automatizados para testes de regressão, a serem executados em um *software* de Gerenciamento de Informações Pessoais (PIM). Este possibilita a troca de informações com dispositivos móveis (telefone celular).

Essa suíte de testes foi desenvolvida para ser executada semanalmente enquanto o *software* testado estivesse na fase de desenvolvimento. Quando estivesse próximo ao lançamento, ele seria executado diariamente, pois é nessa fase que os testes se tornam mais intensos.

Contudo, o maior objetivo deste trabalho foi a possibilidade de a suíte de testes ser executada como parte do processo de *build* diário, ou seja, logo após a geração de um *build* do *software* seria executada a bateria de testes automatizados nesse *software*.

Essa agilidade na execução dos testes permite que os defeitos mais críticos sejam logo identificados e relatados aos responsáveis.

Assim, os responsáveis avaliarão se é ou não possível corrigir logo o defeito e gerar uma nova *build*. Com isso, torna-se possível corrigir rapidamente os defeitos encontrados.

Testes de Regressão

A atividade de teste representa a última revisão de especificações, projeto e codificação e tem como foco principal a avaliação da qualidade do produto através das seguintes atividades (RATIONAL, 2003):

- Localizar e documentar defeitos na qualidade do *software*.
- Avisar de forma geral sobre a qualidade observada no *software*.
- Validar as suposições feitas nas especificações de *design* e requisito através de demonstração concreta.
- Validar as funções do *software* conforme projetadas.
- Verificar se os requisitos foram implementados de forma adequada.

Um conceito de teste fortemente usado em testes automatizados é o de teste de regressão que consiste na re-execução de um subconjunto de testes que já foram executados no passado para assegurar que mudanças recentes no sistema não causaram efeitos colaterais, ou seja, não ocasionaram novos defeitos em funcionalidades do sistema que já haviam sido testadas e aprovadas (CABRAL, 2005).

Isto permite com que mudanças necessárias, seja para corrigir falhas (manutenção), seja para adicionar funcionalidades (evolução) possam ser feitas com um certo grau de confiança em que as mesmas não afetarão funcionalidades já existentes de forma não-intencional (CABRAL, 2005).

Testes de regressão repetem entradas ou casos de testes usados em testes de sistemas e devem ser executados toda vez que o código já existente for modificado.

Rational Robot

Os testes desenvolvidos para esse trabalho foram feitos usando a ferramenta *Rational Robot*. Essa ferramenta serve para gerar e executar *scripts* automatizados para aplicações *Web* e *desktop*.

Através do *Robot* é possível criar e editar *scripts* de duas formas (ROBOT, 2003):

· Gravando: É possível gravar todos os passos executados no computador e a própria ferramenta vai gerando o código.

· SQABasic: É possível criar *scripts* usando os comandos da linguagem.

Quando os *scripts* são desenvolvidos usando o SQABasic é possível identificar os nomes dos objetos que o *Robot* acessará através de um componente que ele possui chamado *Inspector*.

Assim, a ferramenta não prende o testador a somente gravar o código, permitindo também desenvolvê-lo (ROBOT, 2003).

Outro componente importante na ferramenta é o *Rational TestManager*.

Através dele é possível ver detalhadamente o resultado de um teste após a sua execução, pois ao término da execução de um *script*, o *TestManager* exibe um *Log*, mostrando o resultado (ROBOT, 2003).

Suíte de testes de regressão

- **Escolha dos Casos de Teste**

Para automatizar os testes de regressão, primeiramente foram escolhidos alguns casos de teste. Essa seleção foi feita a partir dos casos de testes que são executados manualmente logo após a criação da *build*.

Antes de iniciar qualquer projeto de automação é necessário validar se todos os testes estão aptos a serem executados dessa forma.

É muito importante que o teste automatizado possua um resultado confiável e alguns tipos de testes não permitem a garantia dessa confiabilidade.

Os testes de impressão, por exemplo, não possuem uma forma confiável de serem validados, pois é muito simples mandar algo ser impresso, mas é impossível que o *script* verifique se a impressão foi realmente feita. Isso também acontece com os testes que verificam se o papel de parede do celular foi mudado ou se um determinado som está sendo tocado.

- **Estrutura do projeto**

Após a seleção dos testes, foi criado um *Framework* com os métodos padrão que todos os scripts iriam executar, como por exemplo, abrir e fechar o *software* e imprimir informações nos *logs*. Além disso, foi criada uma hierarquia de bibliotecas, para que métodos comuns ficassem em bibliotecas compartilhadas e os métodos mais específicos ficassem nas bibliotecas de seus respectivos componentes.

Também foram criados *scripts* para a preparação do ambiente, ou seja, *scripts* que fornecessem dados para o computador e para o celular, dados estes que seriam usados posteriormente. Esses dados poderiam ser um contato de telefone, uma imagem, um arquivo, enfim, qualquer dado que fosse necessário para os testes.

- **Comunicação entre o *software* e o telefone**

Era muito importante garantir que o *software* estava apto para que os testes fossem iniciados. Esse passo foi o mais importante e fundamental para todo o sucesso do projeto, visto que o *software* em questão não é uma aplicação comum. Ele tem um grande diferencial que é o fato de ter que se comunicar com um dispositivo externo, o telefone celular.

Portanto, um grande esforço foi despendido para garantir que existia uma

comunicação entre o *software* e o telefone. Sem essa garantia vários erros de comando iriam acontecer e os testes não seriam confiáveis. Isso ocasionaria a necessidade de haver um testador acompanhando os testes e também acabaria com o desempenho dos mesmos.

- ***Screenshots* dinâmicos**

Existem casos de teste que são facilmente validados quando feitos manualmente, como os de abrir ou fechar alguma aplicação. No entanto, foi um grande desafio fazer essa validação através dos *scripts*. Para isso, foi criado um método que tira *screenshots* dinamicamente e os salva em um diretório fornecido previamente.

Através desse método foi possível validar se uma aplicação foi aberta ou fechada sem erros, visto que se pode tirar um *screenshot* antes de abrir a aplicação e outro depois de fechá-la. Com essas duas imagens é possível comparar se elas são iguais.

A ferramenta utilizada, o *Rational Robot*, possui este tipo de funcionalidade. No entanto, através do *Robot* não é possível fazer uma validação dinâmica, ele só permite validações estáticas. Sendo assim, existia a possibilidade de gravar uma tela como *baseline* e depois comparar o resultado atual a essa *baseline*, mas se algo mudasse no software, seria necessário atualizar todas as *baselines*. Com a comparação dinâmica que foi criada, este trabalho não existiu.

Um ponto importante na validação dinâmica que foi desenvolvida foi o fato de a captura do *screenshot* ser somente possível para a tela toda, e não para uma janela individual. Por isso, abarra de tarefas do Sistema Operacional poderia fazer com que dois *screenshots* não fossem iguais, pois no momento em que se tira a primeira imagem, pode-se ter alguns programas abertos e uma determinada hora no relógio. No momento em que se tira a segunda imagem, pode-se ter outros programas abertos e um horário diferente do primeiro. Portanto, para solucionar esse problema, o *screenshot* não leva em consideração a barra de tarefas.

- **Cobertura dos *scripts***

No início do projeto, os *scripts* estavam sendo desenvolvidos somente para um celular, mas ao longo do desenvolvimento ficou claro que os resultados dos testes estavam sendo satisfatórios. Com isso, foi decidido aumentar o suporte a outros celulares, dando uma cobertura total de 8 telefones.

O projeto foi concluído com aproximadamente 150 *scripts* que fornecem cobertura para aproximadamente 100 casos de teste, 8 celulares e 8 componentes do *software* testado. A Figura 1 ilustra esses dados.

Cobertura dos Scripts de Teste

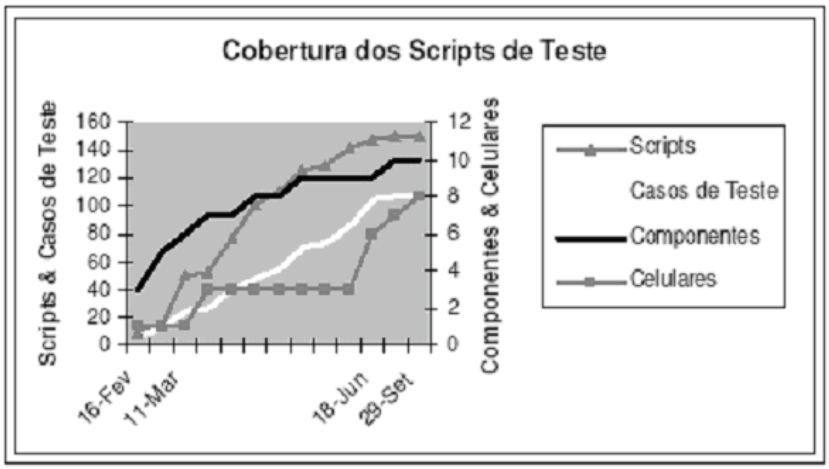

Figura 1 - Cobertura dos Scripts de Teste

Com a automatização, o tempo de execução dos testes foi reduzido. Ao serem executados manualmente eles tinham um gasto médio de 8 horas e com a automatização essas horas foram reduzidas para 2,5.

Além disso, os *scripts* podem ser executados paralelamente em várias máquinas e em diferentes sistemas operacionais.

Conclusão

O ponto fundamental na automação de testes é saber avaliar a viabilidade dos testes, pois não são todos que podem ser automatizados. Existem testes em que não se pode garantir sua execução sem problemas.

Após essa avaliação é de suma importância escolher os casos de teste relevantes para serem automatizados. Em alguns casos, os testes manuais apresentam resultados mais rapidamente do que se a execução fosse feita de forma automatizada. Isso acontece principalmente com testes simples, como abrir ou fechar uma aplicação, onde o tempo para a preparação do ambiente e a verificação do resultado do teste inviabilizam a sua execução automatizada.

Para fornecer mais benefícios na agilidade da execução dos *scripts* é imprescindível que eles sejam robustos, ou seja, que eles possam ser executados sem intervenção humana. Dessa forma será possível executálos em horários fora do expediente normal de trabalho, como por exemplo, à noite, ou em diversas máquinas em paralelo. Com isso, será possível ter os resultados dos testes mais rapidamente e, através de sua análise, os gerentes poderão tomar as decisões a tempo de impedir a entrega de um produto com problemas.

Dando continuidade a este trabalho, uma integração entre os *scripts* desenvolvidos e emuladores de celulares existentes seria um próximo

passo. Usando emuladores, os testes automatizados não teriam tantos problemas de conexão entre o *software* e o telefone.

Outro fato interessante no uso de emuladores é que não se teria problema em executar os *scripts* para telefones que ainda não estão disponíveis ou que ainda não estão com seu *firmware* numa versão estável.

Portanto, através desse trabalho, ficou claro o quanto a atividade de testes e automação dos mesmos é importante no processo de desenvolvimento de *software.* Através dela, os problemas são detectados de forma rápida e antes de o produto chegar ao cliente, podendo dessa forma garantir a qualidade do produto no mercado.

Referências Bibliográficas

(SOMMERVILLE, 2003) SOMMERVILLE, Ian. **Engenharia de Software**. Addison Wesley. São Paulo, 2003.

(MACORATTI, 2004) MACORATTI, José Carlos. Macoratti.net. Disponível em http://www.macoratti.net/tst_sw1.htm. Acessado em 14 de Setembro de 2005.

(CABRAL, 2005) CABRAL, Hermano. **Testes de Software – Notas de Aula – Parte 2**. Manaus, 2005.

(RATIONAL, 2003) **Rational Unified Process**. Version 2003.06.00.65. Copyright 1987 – 2003. Rational Software Corporation.

(ROBOT, 2003) **Rational Robot Users Guide**. Version 2003.06.00. Part

Number 800-026172-000.