Técnicas para Construção de Testes Funcionais Automáticos

Simone Antunes Correia, Alberto Rodrigues da Silva

Resumo — Este artigo apresenta a temática da automatização de testes funcionais na área da Engenharia de Software, iniciando por apresentar alguns conceitos básicos sobre Testes e em seguida mostrando as actuais técnicas para o desenvolvimento de testes automáticos sob sistemas com interface gráfica (GUI). As características e limitações das actuais ferramentas *Capture-Replay* são exploradas bem como diferentes técnicas e abordagens para a construção de testes funcionais automáticos.

Palavras Chave — Testes Automáticos, Testes de Regressão, Testes *Data-driven*, Testes *Keyword-driven*, Ferramentas *Capture-Replay*, Abordagens de Testes.

1 Introdução

A crescente complexidade nos sistemas informáticos juntamente com os métodos de desenvolvimento rápido e incremental – como por exemplo, *Rapid Application Development* [16] e *Extreme Programming* [15], que prometem intervalos de entrega mais frequentes – requerem testes de qualidade que possam ser rapidamente executados sempre que necessário. Em tal cenário os testes manuais são pouco vantajosos, visto que muitos testes são re-executados a cada *release* do sistema.

Os testes automáticos fornecem uma solução neste sentido pois, quando desenvolvidos de forma adequada, serão facilmente executados. Muitos projectos revelam que o conhecimento na área de automação e experiência nos métodos e ferramentas disponíveis, bem como o uso de técnicas que promovam a reutilização e facilidade de manutenção dos testes automáticos, são fundamentais para se obter êxito nesta área [17], [18], [19].

Este artigo contém informações abrangentes na área dos testes automáticos que auxiliarão aqueles que pretendam implantar ou melhorar o processo de testes dentro de uma organização. Através desta leitura, será possível compreender a terminologia usada no âmbito dos testes automáticos, perceber as reais capacidades das ferramentas e conhecer o esforço necessário para construção de testes automáticos de qualidade.

Este artigo apresenta uma análise geral à temática dos testes automáticos organizado da seguinte forma: a Secção 1 introduz a motivação para a apresentação do artigo e descreve a sua organização; a Secção 2 introduz os conceitos básicos sobre testes manuais e testes automáticos usados no âmbito da engenharia de software; a Secção 3 apresenta as características das actuais ferramentas para construção e execução automática de testes funcionais; a Secção 4 apresenta diferentes técnicas para automatização de testes funcionais e finalmente, as considerações finais e de síntese do artigo são apresentadas na Secção 5.

- *Simone Antunes Correia é estudante de mestrado do MEIC- IST/UTL. Trabalha como consultora na área de Testes de Software. E-mail: simone.correia@optimus.pt*
- *Alberto Rodrigues da Silva é Professor auxiliar no Departamento de Engenharia Informática do IST/UTL, investigador sénior no Inesc-ID na área de Sistemas de Informação e consultor em diferentes empresas e instituições. Email: alberto.silva@acm.org*

2. Testes em Engenharia de Software

Podemos definir os testes como uma actividade que tem como objectivo verificar se o software construído está de acordo com sua especificação e se satisfaz as expectativas do cliente e ou utilizador do sistema. Esta definição é uma conclusão a partir do reconhecimento de que a actividade dos testes é parte integrante do processo de Validação e Verificação (V & V) da Engenharia de Software, sendo considerada a técnica dinâmica que exercita a implementação [5].

Nesta secção apresentam-se os conceitos básicos sobre os testes manuais e testes automáticos usados no âmbito da engenharia de software. Introduz-se nomeadamente os tópicos referentes a (1) planeamento de testes; (2) granularidade de testes; (3) abordagens de testes; (4) testes manuais vs automáticos; e (5) automatização da actividade de teste.

2.1 Planeamento de Testes

Tal como referido anteriormente, os testes "exercitam uma implementação", mas é importante salientar que previamente a este exercício os testes devem ser planeados de tal modo que satisfaçam seu objectivo principal que, na opinião de Myers, é de "revelar a existência de defeitos" [7]. Um caso de teste é constituído por um conjunto de **dados de entrada, condições de execução** de uma ou mais operações e **resultados esperados ou dados de saída**, desenvolvidos com um objectivo particular. O desenho dos **casos de teste** e a preparação dos **dados de teste** constituem actividades fundamentais do planeamento dos testes realizadas por um testador.

Visto que o número de casos de teste normalmente é elevado, na maioria das vezes apenas é executado um subconjunto destes casos de testes. Faz parte também da actividade de planeamento determinar quais são os testes mais importantes e que deverão ser executados. Por exemplo, pode ser decidido que, os testes às funcionalidades já existentes

em outra versão do sistema devem ser prioritários aos testes às funcionalidades novas oferecidas na nova versão do sistema. Neste caso, as funcionalidades já existentes que forem escolhidas para serem testadas serão exercitadas pelos chamados Testes de Regressão [6].
Testes de Regressão são definidos como sendo testes aplicados a uma nova versão ou *release* de um sistema a fim de verificar que ele continua a realizar as mesmas funções e da mesma maneira que na versão anterior [6]. Os Testes de Regressão são reutilizados entre diferentes versões do sistema, sofrendo pouca ou nenhuma alteração.

2.2 Granularidade de Testes

Devido à complexidade dos sistemas a actividade de testes deve ser feita ao longo de diferentes estágios. O processo de testes mais utilizado é composto por cinco estágios, como mostramos na Figura 1.

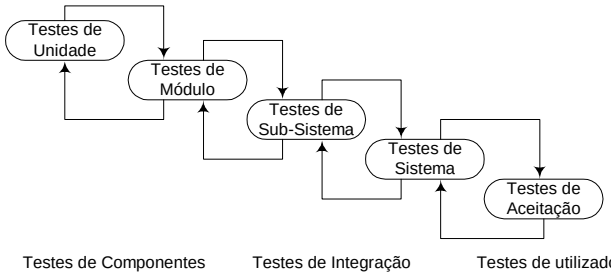

Figura 1 - Visão geral do processo de testes (adaptada de [5])

O processo é iterativo, sendo que a informação, produzida em cada estágio, poderá fluir entre estágios adjacentes. Os estágios do processo de testes segundo definições em [5] são:

Testes de Unidade: Componentes individuais são testados independentemente de outros componentes para certificação de que operam correctamente.
Testes de Módulo: Um módulo é uma colecção de Componentes relacionados tais como classes, tipos abstractos de dados ou um conjunto de procedimentos ou funções. Durante este estágio cada módulo é testado individualmente.
Testes de unidade e de módulo fazem parte do processo de implementação e são da responsabilidade dos programadores que estiveram a desenvolver o componente alvo e ou o componente para testar o componente alvo.
Testes de Sub-Sistema ou Testes de Integração: Esta fase envolve o teste de colecções de módulos integrados em subsistemas. Sub-sistemas podem ser desenhados e implementados independentemente. Um problema comum em grandes sistemas está na integração das interfaces entre os módulos dos sub-sistemas. Estes testes devem portanto concentrar-se no exercício rigoroso de tais interfaces para detecção de possíveis erros.
Testes de Sistema: Os sub-sistemas são integrados para formarem um único sistema. O processo deste tipo de teste irá detectar falhas resultantes das interacções entre subsistemas e componentes de sistema.
Testes de Aceitação: Este é o estágio final do processo de testes antes do sistema ser aceite para uso operacional. O sistema é testado com dados fornecidos pelo utilizador final, ao contrário de dados fictícios ou simulados. Os testes de aceitação revelam principalmente erros e omissões na definição dos requisitos, pois através do uso de dados reais o sistema é exercitado de formas variadas.

2.3 Abordagens de Testes

Testes *Black Box* ou Testes Funcionais e Testes *White Box* ou Testes Estruturais representam as principais abordagens existentes de teste segundo [1]. Qualquer destas bordagens podem ser aplicadas, em princípio, durante qualquer estágio do processo de testes, contudo cada uma delas é preferencialmente aplicável a determinados tipos de componentes e realizáveis por equipas distintas.
Segundo [5], a abordagem funcional por exemplo, é melhor aplicada sob componentes de sistema e realizada por uma equipa de testes, enquanto a abordagem estrutural é melhor aplicada a componentes individuais ou a colecções de componentes dependentes e realizada pela equipa de desenvolvimento.
Através da abordagem de Testes Funcionais, os casos de testes são derivados a partir da especificação do sistema ou componente a ser testado. O sistema é visto como uma caixa fechada e o seu comportamento apenas pode ser derivado através do estudo dos possíveis valores de entrada e dos valores de saída relacionados. Por outro lado, através da abordagem de Testes Estruturais, o testador pode analisar o código e usar o conhecimento da estrutura do componente para derivar os casos de testes. No restante do artigo, quando não explicitada a abordagem de testes, deverá ser assumida como a de testes funcionais.

2.4 Testar versus Automatizar Testes

A qualidade de um caso de teste é descrita através de quatro atributos [3]. O primeiro consiste na capacidade de encontrar defeitos. O segundo refere a capacidade em exercitar mais de um aspecto reduzindo assim a quantidade de casos de teste requeridos. O terceiro e quarto fazem considerações de custo. O terceiro é inferido baseado no custo necessário para a realização do caso de teste incluindo o esforço de desenho, execução e análise dos resultados de teste. O quarto atributo refere o esforço de manutenção necessário sobre o caso de teste a cada alteração do sistema. Estes quatro atributos devem ser balanceados de forma a se ter casos de teste de boa qualidade.
A forma manual ou automática de realização de um teste, não interfere nos dois primeiros atributos citados. A automatização de um caso de teste interfere apenas em quão económico o caso de teste será e que esforço de manutenção será necessário. O esforço de construção e de manutenção requerido para um teste automático é normalmente maior do que para um teste manual equivalente. Mas uma vez construído, o teste automático tende a ser mais económico que o teste manual, o esforço de execução e de verificação de resultados será uma pequena fracção do esforço de construção. Para testes funcionais sob sistemas com interface gráfica foi concluído por Linz e Daigl [11] que, após investimentos iniciais de criação de infra-estrutura, o gasto em testes automáticos representará 40% do gasto com testes manuais.

Visto que o custo original da implementação e o custo da manutenção de um caso de teste automático será diluído a cada execução que seja necessária, os Testes de Regressão são fortes candidatos a serem automatizados.
No entanto, os testes automáticos não são garantia da existência de testes eficazes. A identificação, selecção e desenho dos casos de testes funcionais devem ser feitos por um testador que domine o domínio da aplicação. O responsável que constrói testes automáticos e mantém os artefactos relacionados com o uso de uma ferramenta de execução de testes é chamado de *Test Automator* ou Automatizador de Testes. O automatizador de testes pode ser ou não um testador, devendo sempre ter habilidades em programação. A habilidade do automatizador ditará a qualidade da automação, mas será a habilidade do testador quem ditará a eficácia dos testes realizados.

2.5 Automatização da Actividade de Teste

Nesta secção descrevemos as actividades de teste e suas possíveis formas de automatização. As actividades de teste são normalmente realizadas na seguinte sequência: (1) Identificação, (2) Desenho, (3) Construção, (4) Execução e (5) Comparação, como mostra a Figura 2. As três primeiras actividades fazem parte do planeamento dos testes. Falaremos de cada uma individualmente.

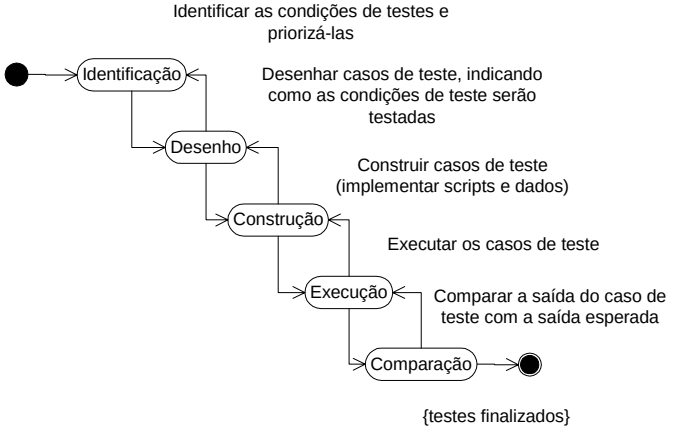

Figura 2 - Actividades de teste (adaptada de [3])

Identificação
Nesta primeira etapa, o testador determinará o que será testado identificando as condições de teste (itens ou eventos) que precisam ser verificados por cada teste. Condições de teste são descrições de circunstâncias que devem ser examinadas. Existem diversas técnicas para uma derivação rigorosa de condições de teste nomeadamente [1]: *equivalence particioning, boundary value analysis, cause-effect graphing*, todas indicadas para uso em uma abordagem de Testes *Black Box* ou Funcional. Algumas técnicas para derivação de condições de teste para uso sob uma abordagem *White Box* é apresentada em [1], como *statement coverage* e *path coverage*.

Desenho
O desenho dos casos de teste determinará como as condições de testes serão testadas. Um caso de teste é um conjunto de testes realizados em sequência e que possuem um objectivo comum que é a razão ou propósito do teste.
Cada caso de teste possui dados de entrada, a informação necessária para a execução do teste e a saída esperada. Os pré-requisitos para realização dos testes, tais como variáveis do ambiente de execução ou estado da base de dados devem ser explicitados para cada caso de teste. A saída ou resultado esperado inclui a criação ou visualização de itens, a modificação ou actualização de itens (em base de dados, por exemplo) ou a remoção de itens. O resultado esperado deve indicar portanto o estado do sistema após a realização da operação sob teste. A actividade de desenho de casos de testes exige um esforço intelectual por parte do testador.
Algumas ferramentas fazem a geração automática dos casos de teste baseadas em código fonte [8], [9], [10] ou especificação formal [21].

Construção
Nesta actividade os casos de teste são transformados em scripts de teste que quando utilizados durante uma execução manual do teste, é também chamado de procedimento de teste. Um procedimento de teste detalha a informação descrita no caso de teste de modo que o testador possa, seguindo as instruções do procedimento, executar e validar a execução de cada teste. Um *script* de teste é normalmente armazenado em ficheiro e escrito em linguagem específica, usada por uma ferramenta de automação da execução do teste. Um *script* de teste pode implementar um ou mais casos de teste. A ferramenta irá processar o script executando as acções por ele descritas.
A preparação dos dados de entrada do teste e do resultado de saída esperado é tarefa fundamental da fase de construção do teste. Quando os resultados de saída forem utilizados para comparação automática através de uma ferramenta, estes devem ser minuciosamente descritos. Uma comparação manual não exige tanto rigor na descrição do resultado esperado que normalmente está incluso dentro do procedimento de teste.

Execução
Nesta actividade o sistema é executado utilizando os *scripts* de teste. Para testes manuais, esta fase pode consistir de testadores a seguirem as instruções existentes em um procedimento de teste. Para testes automáticos as tarefas desta fase podem ser resumidas apenas no lançamento do *script* de teste correspondente. Os dados de entrada podem estar no *script* de teste ou separado em ficheiros ou em base de dados. A ferramenta irá executar o *script*, efectuando as acções que o testador efectuaria manualmente. Existem diferentes ferramentas para execução automática de testes e diferentes técnicas para criação de tais *scripts* como veremos apresentados em 4.

Comparação
Os resultados obtidos do sistema sob testes são usados para determinação do resultado do teste. Existem dois possíveis resultados: positivo ou negativo. A verificação do resultado obtido pode ser feita com confirmação informal do que o testador espera ver ou pode ser um comparação rigorosa e detalhada entre o resultado obtido e a resultado esperado (determinado durante a fase de construção do teste). Alguns resultados podem ser comparados ainda durante a execução do teste (uma mensagem que deve ser enviada ao ecrã), mas, por exemplo, resultados que fazem modificações

de registos em base de dados podem apenas ser comparados depois que a execução do teste é finalizada. Um teste automático pode utilizar os dois métodos de comparação.

3. FERRAMENTAS PARA CONSTRUÇÃO E EXECUÇÃO DE TESTES AUTOMÁTICOS FUNCIONAIS

As ferramentas para construção de testes automáticos funcionais permitem a realização dos testes de uma forma não assistida.

Na secção seguinte iremos destacar as ferramentas do tipo *capture-replay* fazendo uma descrição de suas funcionalidades, recursos, limitações e contextos de utilização. A ferramenta comercial WinRunner [14] será usada para a exemplificação de alguns aspectos descritos.

3.1 Funcionalidades

As ferramentas *capture-replay* ou gravação-execução que daqui por diante abreviaremos por ferramentas GE, são ferramentas que permitem a criação, execução e a verificação dos resultados de testes automáticos para sistemas com interface gráfica standard. Na indústria informática, são as ferramentas mais conhecidas para a realização de testes automáticos.

As ferramentas GE possuem duas funções principais de utilização, a função de gravação e a função de repetição. Através da primeira, todos os objectos visualizados no sistema a ser testado são registados e toda a sequência de interacções realizadas sob estes objectos são registados/gravados em ficheiro. Este ficheiro ou *script* de teste, como passa a ser chamado, contém todos os movimentos do rato e teclado realizados, todas as entradas inseridas, as opções escolhidas e o resultado obtido. Quando interrompido o modo de gravação, a ferramenta pára de registar as acções no *script*.

Através da função de repetição, qualquer *script* escrito na linguagem da ferramenta poderá ser executado, incluindo aquele criado pela função de gravação. A execução de um *script* que tenha sido gravado será uma repetição fiel de todas as acções realizadas aquando da gravação.

O resultado obtido durante a gravação pode ser considerado o resultado esperado do teste e será comparado com os resultados obtidos durante as subsequentes execuções automáticas como forma de determinar o resultado final do teste.

3.2 Recursos Utilizados

Existem duas principais formas usadas pelas ferramentas GE para o reconhecimento dos elementos de uma interface gráfica da aplicação sob testes. A primeira é orientada pela posição das coordenadas do elemento gráfico no ecrã. A segunda, mais flexível, reconhece os elementos da interface como objectos gráficos, possuidores de propriedades que determinam o seu aspecto e comportamento.

Para a ferramenta WinRunner [14], que realiza testes sob aplicações com interface Windows ou Web, a lista das propriedades físicas juntamente com os respectivos valores de atribuição, usados para identificação de cada objecto gráfico, formam o que é chamado de "**descrição física**" do objecto [22]. No entanto, os *scripts* de teste referenciam os objectos utilizando outro identificador, chamado de "**nome lógico**". A relação entre o descritor físico e o nome lógico é mantida em ficheiros chamados de GuiMap[22]. Esta solução, permite que os *scripts* mantenham-se sintaticamente válidos mesmo quando há alteração nas propriedades físicas dos objectos, passando a ser necessário apenas alteração no ficheiro GuiMap.

A maioria das ferramentas GE permitem a criação de testes que interajam com objectos gráficos e janelas criados com o standard *Microsoft Foundation Class Library* (MFC). Objectos e janelas criados usando tecnologias diferentes por exemplo, *Java Foundation Class Library* podem ou não ser suportados pela ferramenta envolvida [13]. No caso da ferramenta WinRunner existem *add-ins* para o tratamento de aplicações Java, PowerBuilder, Visual Basic e aplicações Web [12]. Caso uma ferramenta GE não consiga identificar o tipo do objecto gráfico, como último recurso a ferramenta GE pode sempre reconhecer objecto pelas suas coordenadas espaciais do ecrã.

Muitas ferramentas GE, incluindo WinRunner, possuem funções próprias para a realização de *queries* SQL sob qualquer base de dados que suporte a interface ODBC. Tal funcionalidade permite inclusive que a verificação do resultado não seja feita apenas baseado no que é visualizado no ecrã, mas sim no que foi realmente alterado em base de dados.

A possibilidade de permitir a criação de bibliotecas de funções reutilizáveis é uma funcionalidade bastante desejável e existente em muitas ferramentas. Além de permitir a criação de bibliotecas próprias algumas ferramentas permitem o acesso a bibliotecas externas através de chamadas a ficheiros .dll.

As linguagens de scripts fornecidas pelas ferramentas GE normalmente são proprietárias e interpretadas. A ferramenta WinRunner fornece uma linguagem estruturada, semelhante a C, chamada TSL (*Test Script Language*) [22].

3.3 Limitações

As actuais ferramentas comerciais GE são vendidas como sendo uma solução fácil e auto-suficiente para a realização de testes automáticos, no entanto a experiência mostra que os *scripts* gravados possuem algumas limitações que inviabilizam a sua utilização como única forma de criação de testes. Abaixo mostraremos algumas das fragilidades dos *scripts* gravados, designadamente:

- Os *scripts* são pobremente estruturados. Os *scripts* gerados pelo modo de gravação resultam em *scripts* extensos onde todas as acções efectuadas são registadas. Muitas destas acções poderiam ser reutilizadas por outros testes mas são repetidamente gravadas em cada novo *script* de teste gravado.
- Inexistência de mecanismos de reutilização de código. As acções (código), os dados de entrada e de resultado esperado ficam todos juntos no *script* criado pelo modo de gravação. Isto não permite que os *scripts* sejam escaláveis e reutilizados pois os dados estão hard-coded no *script*. Muitas vezes o que difere um caso de teste de outro são os dados

por ele processados, enquanto as acções são as mesmas.

Todas estas limitações podem ser ultrapassadas se, tal como acontece na programação, utilizarmos as técnicas apropriadas para construção dos *scripts*. Na Secção 4 veremos algumas formas de se desenvolver *scripts* com baixo custo de manutenção e mais vantajosos num desenvolvimento a longo prazo.

3.2 Contextos de Utilização

As ferramentas GE são fundamentalmente usadas para a realização de testes funcionais, de aceitação, sob sistemas com interface gráfica.

No entanto, se quisermos realizar testes automáticos sob formatos não gráficos, como por exemplo testes a *stored procedures PL-SQL*, a *WebServices*, aplicações servidor ou a um componente através de uma dll, a única solução possível passa pela criação de uma camada de interface gráfica standard simples, que possibilitará a invocação dos serviços a serem testados. Esta interface abstrairá as características da interface não tratada directamente pela ferramenta, possibilitando a invocação dos serviços a serem testados. Através da utilização deste recurso, podemos dizer que as ferramentas GE podem ser aplicáveis para execução e comparação automática de testes em todos os níveis: unidade, módulo, integração, sistema ou aceitação.

4. Técnicas para Construção de Scripts de Teste

As técnicas para construção de *scripts* de testes automáticos são similares às técnicas de programação. Os *scripts* de testes automáticos contêm dados e instruções para a ferramenta de teste. A minimização do esforço de manutenção dos *scripts* só é conseguida através de um investimento na construção dos *scripts*. Em [3], os autores consideram a existência de cinco técnicas para criação de *scripts*, que se refere de seguida.

Scripts Lineares

Um *script* linear é aquele obtido a partir da gravação feita por uma ferramenta GE. É uma rápida forma de começar a construir *scripts* de testes automáticos, pois não requer conhecimento da linguagem oferecida pela ferramenta. No entanto, estes *scripts* não são úteis num plano a longo prazo. Normalmente, possuem informação excessiva e repetida, dados registados junto às acções (hard-coded) e estão muito associados a particularidades do sistema na altura da gravação o que os torna bastante vulneráveis a mudanças no sistema a ser testado. A criação de testes automáticos através de *scripts* gravados não é portanto uma boa prática.

Scripts Estruturados

Tal como as linguagens de programação estruturadas estes *scripts* usam estruturas de controlo como "If" e "Loop", o que garante uma flexibilidade não existente nos *scripts* lineares. Na escolha da ferramenta convém verificar, entre outras, a capacidade das suas linguagens no que se refere às instruções de controlo.

Scripts Partilhados

São *scripts* que podem ser usados por mais de um caso de teste. Os *scripts* partilhados podem ser específicos a uma aplicação ou independente de aplicação. Uma vez identificado um conjunto de acções úteis para mais de um teste, um *script* partilhado deve ser criado e disponibilizado para invocação a partir de qualquer outro teste. As informações variáveis devem ser passadas como parâmetro, tal como acontece na invocação de uma função na programação estruturada.

Scripts *Data-driven*

São *scripts* mais abrangentes que lêem entradas de testes ou resultados esperados a partir de um ficheiro de dados ou tabela de dados evitando termos dados *hard-coded* no próprio *script*. Além disto esta técnica permite que novos testes sejam adicionados mais facilmente, uma vez que em alguns casos a existência de novos testes pode ser expressa pela inclusão de novas entradas na tabela de dados, sem nenhuma alteração no *script* de controlo. Os testes podem ser adicionados sem a necessidade de alteração no código do *script*. Em adição à entrada de teste, o resultado esperado também pode ser removido do *script* e colocado no ficheiro de dados, uma vez que o resultado esperado está directamente associado com a entrada do teste.

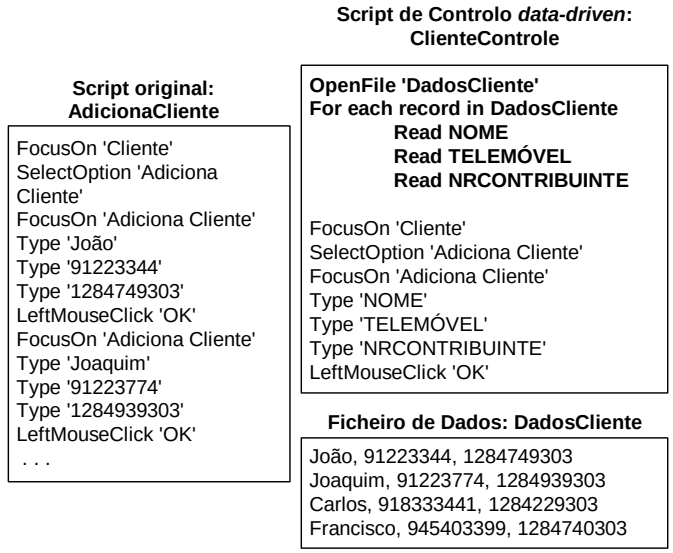

Figura 3 - O script original AdicionaCliente, implementado pela técnica data-driven (adaptada de [3])

Muitas ferramentas GE encorajam a utilização desta técnica fornecendo mecanismos para armazenamento de dados de entrada em ficheiro texto e atribuição dos mesmos, durante a execução, a variáveis do *script*.

Como uma limitação desta técnica podemos citar que os scripts *data-driven* requerem que o desenho do teste seja feito na linguagem de automação da ferramenta. Desta forma, todos os envolvidos com o desenvolvimento de testes automáticos ou execução automática de testes precisam ser conhecedores do ambiente e da linguagem de programação da ferramenta de automação. A Figura 3 apresenta um exemplo de utilização desta técnica para testes a um sistema de facturação. Os testes apresentados fazem a inserção de clientes, onde os dados requeridos são: nome, telemóvel e número de contribuinte.

Scripts *Keyword-driven*

Na técnica *data-driven* a navegação e acções realizadas são as mesmas para cada caso de teste e o conhecimento lógico sobre os testes está distribuído no ficheiro de dados e no *script* de controlo e precisam ser sincronizados. A técnica *keyword-driven* combina a técnica *data-driven* com a possibilidade de especificar os casos de teste de forma menos detalhada, tal como é feito quando estamos a descrever um caso de teste manual.

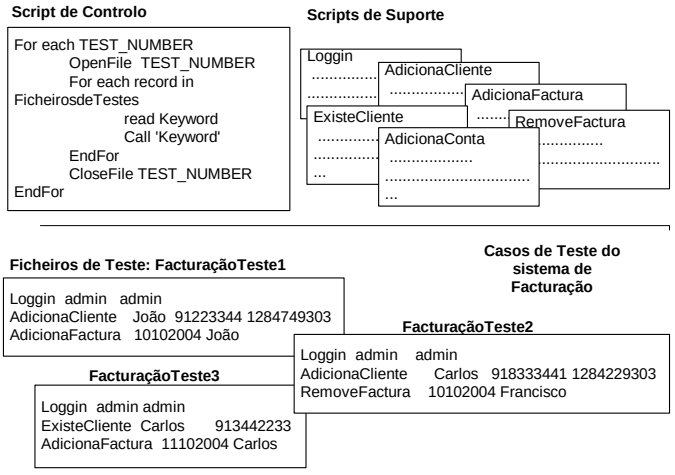

Figura 4 – Aplicação da técnica keyword-driven (adaptada de [3])

Esta técnica expande o ficheiro de dados da técnica anterior de forma a torná-lo uma descrição dos casos de teste que queremos automatizar, utilizando um conjunto de palavras chave (*keywords*). Este novo ficheiro que é chamado de ficheiro de testes descreve apenas o que o caso de testes faz, mas não a forma como é feito. O *script* de controlo tem habilidade de interpretar estas *keywords*, as quais estão implementadas fora do *script* de controlo. Esta separação na implementação das *keywords* requer um nível adicional de implementação técnica, que é feito através dos chamados *scripts* de suporte. Temos portanto três estruturas básicas que são o *script* de controlo, os ficheiros de teste e os *scripts* de suporte.

Esta técnica permite usar o conhecimento de um testador experiente no domínio da aplicação para o desenho ou construção dos casos de testes em ficheiros de teste e o conhecimento técnico de um automatizador nas ferramentas e linguagens existentes para a construção dos *scripts* de suporte, utilizando assim perfis distintos de testadores para a construção de testes automáticos mais efectivos e robustos.

A Figura 4 apresenta a utilização desta técnica para o exemplo de um sistema de facturação, onde os testes pretendem verificar a realização de operações básicas, como adicionar/remover factura e adicionar/remover cliente. As palavras chave AdicionaFactura e RemoveFactura, são invocadas em diferentes testes ou ficheiros de teste, mas estão implementadas apenas nos *scripts* de suporte correspondentes.

5. CONSIDERAÇÕES FINAIS

Concluímos este artigo com a apresentação da Figura 5 que sintetiza os conceitos e classificações apresentados neste artigo sobre os testes de sistemas informáticos.

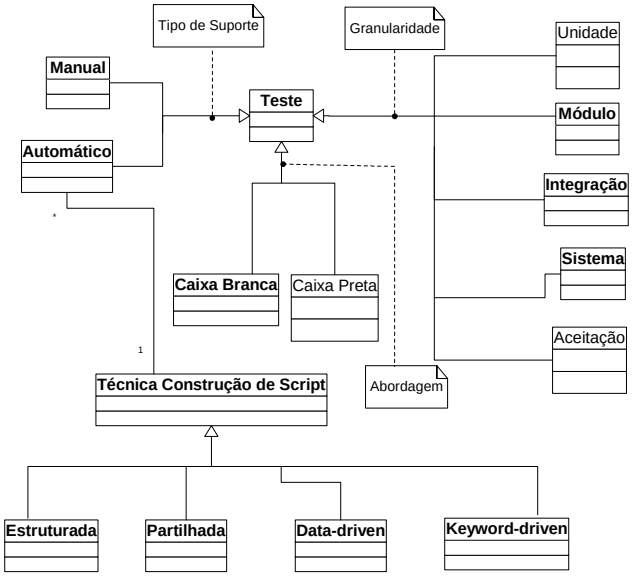

Figura 5 - Visão geral dos tipos de testes

Vista como actividade que promove o aumento na qualidade do software, a realização de testes de acordo com uma metodologia tem vindo a se tornar cada vez mais frequente na indústria informática. Vimos neste artigo que esta actividade deve ser realizada segundo um processo bem definido que valorize a qualidade do desenho dos testes independente do tipo de suporte possível: manual ou automático.

Neste artigo, apresentamos os diferentes tipos de testes segundo sua granularidade e as principais abordagens de teste existentes. Indicamos como cada uma das actividades de testes pode ser realizada automaticamente, salientando e discutindo o contexto onde a automatização é mais indicada.

Apresentamos as funcionalidades das ferramentas que permitem a execução e comparação automática dos testes, bem como suas limitações e contextos de utilização. Diferentes técnicas para a construção de testes automáticos foram apresentadas, tendo sido salientadas as mais evoluídas *data-driven* e *keyword-driven*, que permitem a criação de testes automáticos reutilizáveis e flexíveis.

Um procedimento favorável ao sucesso da automação é tratá-lo como um projecto de desenvolvimento de software [23]. Neste sentido, sugere-se a criação de uma infra-estrutura de testes automáticos reutilizáveis e de fácil manutenção, com um objectivo de sucesso a longo prazo. Apenas desta forma poderão ser constatadas as vantagens da automação.

Recomendamos que a realização dos testes automáticos seja feita de forma planeada, a pensar não apenas no sistema actualmente a ser testado, mas tendo em consideração que, com algum esforço adicional, os *scripts* poderão ser usados,

por diferentes *releases* do mesmo sistema, e também por diferentes sistemas, com menor esforço de concepção e desenvolvimento.

A partir da constatação de que muitas actividades de testes são comuns a diferentes sistemas, pensamos que, um desafio para o futuro está na concepção de arquitecturas de testes e na implementação de testes genéricos. Tais testes, mediante configuração específica do sistema alvo, nomeadamente o modelo de interface gráfica e de estrutura de dados, poderiam ser utilizados eficientemente por distintos sistemas.

REFERÊNCIAS

[1] E. Kit. *Software Testing in the Real World: Improving the Process*. Addison-Wesley, 1995.

[2] Bodgan Bereza, Jarocinski. Tools and Automation in a Shoestring. *Workshop Eurostar*. Estocolmo, 2001.

[3] M.Fewster e D.Grahm. *Software Test Automation*. Addison-Wesley, 1999.

[4] H.Buwalda, D.Janssen, I.Pinkster. *Integrated Test Design and Automation: Using the TestFrame Method*. Addison-Wesley, 2002.

[5] Sommerville. *Software Engineering*. Addison-Wesley, 2000.

[6] Share Lawrence Pfleeger. *Software Engineering: Theory and Practice*. Prentice Hall, 2001.

[7] Glenford J.Myers. *The Art of Software Testing*. New York: John Wiley and Sons, 1979.

[8] IPL, www.iplbath.com (AdaTEST, Cantata, Cantata++)

[9] LDRA Ltd, www.ldra.com (LDRA Testbed, geração automática de ambiente de testes - harness, stubs e drivers)

[10] Testwell, www.testwell.sci.fi/homepage.html (ferramentas de teste para C, C++ e Java)

[11] Tilo Linz, Matthias Daigl. White Paper*: How to Automate Testing of Graphical User Interfaces*. Alemanha, 1998. http://www.imbus.de/forschung/pie24306/gui/aquis-full_paper-1.3.html.

[12] Horwath, Green & Lawler. White Paper: *SilkTest and WinRunner Feature Descriptions*, 2000.

[13] Elisabeth Hendrickson. *Making the Right Choice*. Revista StqeMagazine, Maio/Junho 1999. http://www.qualitytree.com/feature/mtrc.pdf

[14] WinRunner information. http://www.mercury.com/us/products/quality-center/functional-testing/winrunner/

[15] Kent Beck. *Extreme Programming Explained*. Addison-Wesley, 2000.

[16] Steve Mcconnell. *Rapid Development*. Microsoft Press, 1996.

[17] JimDougherty, Keith Haber. Test Automation: Reducing Time to Market. *International Conference on Software Testing, Analysis & Review*, 2002.

[18] Tilo Linz. *Case Study: IMBUS GmbH. Automated Testing of Graphical User Interfaces*, 1998. www.esi.es/VASIE/Reports/All/24151/gui-test1.pdf

[19] Elisabeth Hendricson. White Paper: *The Difference between Test Automation Failure and Success, 1998*. www.qualitytree.com/feature/dbtasaf.pdf

[20] Cem Kaner. Improving the Maintainability of Automated Test Suites. *Quality Week Conference, 1997*

[21] ADL Translation System

[22] WinRunner User's Guide Versão 7.6, Mercury Interactive Corporation.

[23] B.Pettichord. Seven Steps to Test Automation Sucess. *STAR West Conference*, 1999. Versão revisada em 2001

Simone Antunes Correia Licenciatura em Ciência da Computação pela Universidade Federal de Pernambuco. Estudante de mestrado do MEIC-IST/UTL. Trabalha actualmente como consultora na área de Testes de Software.

Alberto Rodrigues da Silva Doutoramento e mestrado em Engenharia Informática e Computadores pelo IST/UTL e licenciatura em Engenharia Informática pela FCT/UNL. Professor auxiliar no Departamento de Engenharia Informática do IST/UTL, investigador sénior no INESC-ID na área de Sistemas de Informação e consultor em diferentes empresas e instituições.