AUTOMATIZAÇÃO DE CASOS DE TESTE COMO PROCESSO DE MELHORIA DA QUALIDADE DO SOFTWARE: O CASO DA APLICAÇÃO E-LEARNING ISUPAC3 NO ISUTC

Elton Sixpence*[1], Pedro Adão[2] e Cameron Smith***[3]**
[1]ISUTC – Maputo, Moçambique, [2]IST – Lisboa, Portugal, [3]ENGCO – Maputo, Moçambique
**E-mail*: elton.sixpence@isutc.transcom.co.mz
***E-mail*: pedro.adao@ist.utl.pt
****E-mail*: cameron.smith@engco.co.mz

RESUMO: *Apesar das aplicações Web terem crescido em popularidade nos últimos anos, o estudo das ferramentas que suportam o teste das suas Interfaces com o utilizador continua a ser até agora uma área pouco explorada. Estas apresentam alguma similaridade com outras tecnologias de Interfaces com o Utilizador no entanto um aspecto que necessita ainda de mais evolução é o estudo duma* framework *que permita automatizar o processo de teste das mesmas.*

Neste trabalho é utilizada a ferramenta Sahi*, desenhada para suportar testes de interfaces baseadas em tecnologias AJAX, para a implementação de testes automatizados a uma aplicação Web. O protótipo é utilizado sobre uma Aplicação de* e-learning *desenvolvida no ISUTC chamada ISUPAC3.*

1. MOTIVAÇÃO

Com a finalidade de apoiar o processo de avaliação contínua em curso no ISUTC (Instituto Superior de Transportes e Comunicações), que inclui a realização de um elevado número de provas de avaliação ao longo do semestre, foi iniciada em 2003 a criação da ISUPAC3, Pacotes de Auto-Aprendizagem e Avaliação Computarizados do ISUTC, uma Aplicação Web de e-Learning [Oliveira, 2008].

Quando foi iniciado o desenvolvimento da ISUPAC3, uma única pessoa era capaz de executar manualmente todos os casos de teste em apenas dois dias mas, devido ao aumento das funcionalidades e consequente aumento exponencial dos casos de teste, tal facto é actualmente impossível. Esta impossibilidade deriva não só da maior complexidade dos actuais casos de teste mas também do tempo necessário para a execução dos mesmos.

Aliado à maior complexidade do actual processo de testes, o tempo disponível para a sua execução nem sempre é o ideal devido a derrapagens do processo de desenvolvimento. Este facto leva a que muitas vezes sejam ignorados alguns casos de teste o que torna a aplicação susceptível a falhas e defeitos em quantidade não desprezível, e implica uma reduzida qualidade da aplicação em produção [Sixpence, 2010]. Qualquer solução para resolver este problema passa necessariamente pela execução dos testes de uma forma mais expedita.

Pela natureza repetitiva dos testes de regressão, pela necessidade de reduzir o esforço manual do processo de testes, e consequentemente para o aumento da qualidade da ISUPAC3, surgiu a ideia de utilizar a ferramenta *Sahi* para a execução automática dos testes de regressão e de funcionalidade bem como para a definição de um padrão de escrita dos *scripts* de teste. Apesar deste processo apresentar uma melhoria significativa em termos de tempo de execução, este requer um elevado investimento que apenas será compensado a médio prazo [Sixpence, 2010].

2. DESCRIÇÃO DO PROBLEMA

A actividade de teste de Software comporta quatro etapas, nomeadamente i) *setup* do ambiente de testes, ii) execução dos casos de teste, iii) análise do resultado da execução dos casos teste, e iv) manutenção dos casos de teste [M. Pezze, 2008].

Dentre as quatro etapas, a de execução apresenta processos repetitivos, isto é, para cada Versão do Software devem ser executados casos de teste para as novas funcionalidades bem como para as já existentes, apesar de um ou outro sofrerem actualizações. Por outro lado, mesmo que o Software esteja em Produção, recomenda-se que sejam feitos testes das funcionalidades já testadas, os chamados testes de regressão.

A ISUPAC3 apresenta um elevado número de funcionalidades pelo que a execução de todos os casos de teste é um processo demasiado complexo para ser executado manualmente, mostra-se cansativo para os testadores, e a probabilidade de serem cometidos erros é relativamente elevada. Apesar de se realizarem testes de funcionalidades durante o desenvolvimento de cada versão, continuam a ser detectadas falhas e defeitos na ISUPAC3 já em produção. Para além disso, o processo de execução manual dos testes apresenta uma tendência para aumentar os seus custos devido ao crescente número de casos de teste resultantes da evolução da ISUPAC3, pelo que queremos encontrar uma solução que minore tais custos.

Tomando em consideração a existência de técnicas que podem ser utilizadas para automatizar o processo de execução de testes e consequentemente a redução dos recursos despendidos na etapa de execução dos testes, este trabalho tem como objectivos específicos responder às seguintes questões:

1. Automatizando os testes de funcionalidade e de regressão da ISUPAC3, até que ponto é possível reduzir os recursos associados à execução destes?
2. Que tipo de defeitos e/ou falhas o teste automatizado será capaz de detectar?
3. Que mudanças deverão ser feitas ao nível da Célula de Testes do ISUTC para acomodar o sistema automatizado de testes?
4. Quais são os outros ganhos obtidos com o processo de automatização dos testes da ISUPAC3?

3. ESTADO DA ARTE

3.1. Técnicas de teste funcional

O teste pode ser definido como uma actividade que tem por objectivo verificar se o Software produzido 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 de teste é parte integrante do processo de Validação e Verificação (V&V) da Engenharia de Software, sendo considerada a técnica dinâmica que exercita a implementação [Sommerville, 2000].

A crescente complexidade dos sistemas informáticos juntamente com os métodos de desenvolvimento rápido e incremental – como por exemplo, *Rapid Application Development* e *Extreme Programming* (Beck, 2000) que prometem intervalos de entrega mais curtos, 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 Versão do sistema.

Os testes automáticos fornecem uma solução neste sentido pois, quando desenvolvidos de forma adequada, serão facilmente executados. [Dougherty, 2002].

3.2 Granularidade de Teste

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

O processo é iterativo, sendo que a informação produzida em cada estágio poderá fluir para estágios adjacentes. Segundo Sommerville os estágios do processo de teste são descritos da seguinte forma:

Testes de Unidade: Componentes individuais são testados independentemente de outros componentes para certificação de que operam correctamente. Nestes testes procuram-se falhas em pequenas unidades do sistema, como por exemplo o comportamento dum método pertencente a uma certa classe.

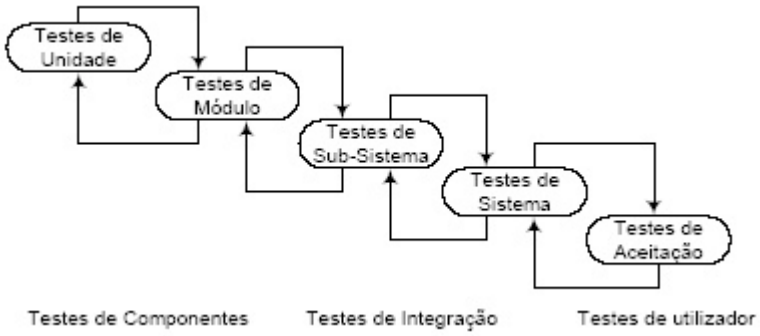

Figura 1 - Processo de testes de Software
Fonte: [Sommerville, 2000]

Testes de Módulo: Um módulo é uma colecção de componentes relacionadas 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 Software. É a este nível que são detectadas as falhas de interacção de unidades que constituem um módulo.

Testes de Subsistema ou Testes de Integração: Esta fase envolve o teste de colecções de módulos integrados em Subsistemas. Estes podem ser desenhados e implementados independentemente. Um problema comum em grandes sistemas está na integração das interfaces entre os módulos dos subsistemas. Estes testes devem portanto concentrar-se no exercício rigoroso de tais interfaces para detecção de possíveis erros e/ou falhas que possam resultar do processo de integração.

Testes de Sistema: Os subsistemas 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 de o 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.

3.3. Planeamento de Teste de Software

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 da actividade de teste realizada por um analista de teste.

Visto que o número ideal de casos de teste é normalmente elevado, na maioria das vezes apenas é executado um subconjunto destes. Faz parte também da actividade de planeamento determinar quais são os testes mais importantes e que por isso deverão ser executados. Por exemplo, pode ser decidido que os testes às funcionalidades já existentes numa versão anterior do sistema devem ser prioritários face aos testes das novas funcionalidades 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 [Pfleeger, 2001].

3.4 Abordagens de Teste

Testes *Black Box,* ou Testes Funcionais, e Testes *White Box* ou Testes Estruturais, representam as principais abordagens de teste existentes [Kit, 1995]. Qualquer destas abordagens pode ser aplicada durante qualquer estágio do processo de teste, contudo cada uma delas é preferencialmente aplicável a determinados tipos de componentes e realizáveis por equipas distintas.

A abordagem funcional, por exemplo, é melhor aplicada sobre 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 teste 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 da relação entre os possíveis valores de entrada e os respectivos valores de saída. Por outro lado, através da abordagem de Testes Estruturais, o analista de testes pode analisar o código e usar o conhecimento da estrutura do componente para derivar os casos de teste [Sommerville, 2000].

3.5 Automatização da Actividade de Teste

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. As outras duas dizem respeito à execução dos testes.

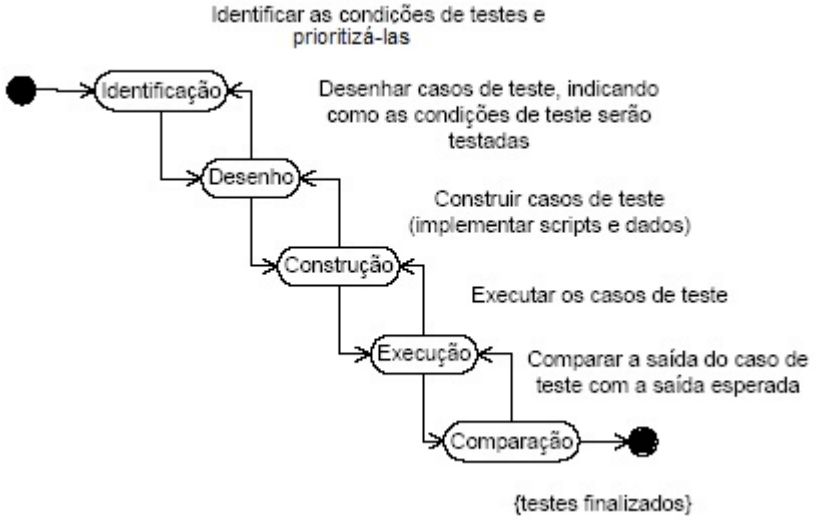

Figura 2 - Etapas de automatização de testes
Fonte: [Fewster, 1999]

3.6 Ferramentas de Automatização de Testes

Um método comum nas ferramentas de testes de caixa-preta é o método *Capture/replay* que pressupõe a existência de uma infra-estrutura que suporta a captura e reprodução dos cenários do utilizador. Durante os testes das Aplicações Web as interacções do utilizador com o Servidor podem ser simuladas gerando grafos de eventos provocados pelo computador e que estejam associados à Interface da Aplicação. Uma das formas de implementar este método por forma a obter um resultado é gravar as interacções do utilizador com a Aplicação e depois reproduzi-las durante a fase de execução do teste de regressão.

Apesar de existirem muitas ferramentas de testes que podem ser empregues no teste de Aplicações Web, a selecção destas não é um processo trivial, pois existem muitas tecnologias para o desenvolvimento de Aplicações Web e nem todas as ferramentas de teste suportam todas as tecnologias. Daí a necessidade de terem sido estudadas algumas ferramentas e com base no estudo foi escolhida a ferramenta compatível com a ISUPAC3.

3.6.1 Watir

O *Watir*[1] é uma biblioteca *open source* BSD (*Berkeley Software Distribution*) utilizada para automatizar o teste de Aplicações executadas num *browser* através da escrita de *scripts* de teste de fácil interpretação e manutenção. Permite manejar o *browser* como um utilizador, como por exemplo seleccionar *links*, preencher formulários, pressionar botões, de entre outras operações. Depois de executados os *scripts,* é possível verificar os resultados das acções executadas tal como o texto ou acção esperada na página Web. Trata-se duma biblioteca da família *Ruby* que suporta aplicações independentemente da tecnologia que foi utilizada para o seu desenvolvimento. O exemplo que se segue mostra o quão simples é escrever um *script* de teste através do *Watir*:

```
require 'watir' // permite ao Watir manipular o Internet Explorer
browser=Watir::Browser.new //inicialização do browser
browser.goto("http://www.transcom.co.mz") //abertura da página Web
browser.checkbox(:value => "Ruby").set
browser.checkbox(:value => "Python").set
browser.checkbox(:value => "Python").clear
```

Executando o exemplo acima, será aberta uma sessão do *browser Internet Explorer,* visualizado o *site* http://www.transcom.co.mz, serão seleccionados os *checkboxes* com o rótulo *Ruby e Python*, e finalmente desactivado o *checkbox* com o rótulo *Python*.

Existem algumas motivações que tornam o *Watir* uma ferramenta muito utilizada no teste de funcionalidade de Aplicações Web nomeadamente i) o suporte de aplicações Web independentemente da linguagem em que foram desenvolvidas, ii) ser uma ferramenta *open source* e sem custos de licença, e iii) a existência de uma comunidade muito activa trabalhando sobre esta. Apesar destas vantagens, apresenta alguns inconvenientes como é o caso da dificuldade que existe em escrever *scripts* de teste para Aplicações baseadas em AJAX. As especificidades do AJAX não encontram um tratamento eficiente a nível do Watir. Outra desvantagem apresentada por esta ferramenta é o facto de não permitir a criação de *scripts* de teste por gravação, o que torna este processo dependente de excessiva intervenção manual.

3.6.2 Sahi

[1] http://watir.com

O Sahi[2] é uma ferramenta *open source* de automação de teste de aplicações Web. A criação de *scripts* de teste é feita através de um Servidor *Proxy* posicionado entre o *browser* e o Servidor da Aplicação alvo de teste. Os *scripts* de teste são gerados com recurso à técnica de gravação numa Interface gráfica e possibilita a posterior edição dos *scripts* gerados permitindo assim melhorar a qualidade dos mesmos. O facto de o Sahi utilizar um *Proxy* para gerar os *scripts* de teste permite que este suporte teoricamente todos os *browsers* e garante a independência do Sistema Operativo. Um aspecto importante encontrado no *Sahi* é o suporte completo do teste de Aplicações Web dinâmicas e *AJAX* dado que interpreta o *Javascript* e possui mecanismos para lidar com a variação dinâmica da estrutura do DOM (Document Object Model).

O Sahi apresenta-se com uma linguagem de programação similar ao *Javascript*, seguem-se dois exemplos de utilização da linguagem de *scripting* do Sahi:

a) *Statements* são linhas normais de código e terminam com um ponto e vírgula, por exemplo:

```
_click(_link("Login"));
```

b) Declaração de variáveis: Todas as variáveis começam pelo símbolo $ como por exemplo:

```
var $variableName; // declaração
$variableName = value; // atribuição
```

No entanto o *Sahi* apresenta algumas desvantagens tais como o facto de não suportar páginas Web desenhadas com recurso a *Frames*, não carregar páginas de vários domínios, e não permitir verificar ficheiros carregados através do browser apesar de suportar o carregamento dos mesmos.

3.6.3 Badboy

O Badboy[3] é uma aplicação proprietária, desenhada para automatizar testes de *Interfaces Web* utilizando o *browser Microsoft Internet Explorer*. Este aspecto é a grande desvantagem apresentada por este produto, pois a maior parte das aplicações são desenhadas para serem utilizadas em múltiplos *browsers*. Por outro lado o *Badboy* é extensível pois permite que sejam agregadas novas funcionalidades. A Aplicação *Badboy* permite a gravação de scripts e automatização da execução de testes, e possui opções para execução a partir da linha de comandos o que possibilita o arranque automático do teste.

O resultado dos testes é fornecido a pedido, isto é, nos scripts adicionam-se comandos especiais para o efeito, e é ainda possível guardar os *logs* no sistema de ficheiros ou então enviá-los por e-mail.

O Badboy permite a integração com uma outra ferramenta denominada WTM (*Wave Test Manager)*, que pode ser configurada para actuar como controlador do Badboy. O WTM permite aos utilizadores a execução de *scripts* de teste em diferentes computadores ao mesmo tempo o que permite realizar testes simultâneos em diferentes plataformas. Todavia esta funcionalidade é questionável, dado que o Badboy é apenas capaz de realizar testes em diferentes versões do *browser Internet Explorer* ao mesmo tempo. O gestor de testes utiliza uma interface Web o que permite centralizar a manutenção. O WTM auxilia a integração da execução de testes no ciclo de desenvolvimento.

3.6.4 Selenium

---

[2] http://sahi.co.in

[3] http://badboy.com.au

É um projecto *open source* com uma larga base de suporte. Possui operações flexíveis que permitem manipular os elementos do Interface de Utilizador de uma Aplicação Web para além de comparar os resultados obtidos com os esperados.

A utilização de comandos no Selenium consiste em digitar o comando seguido de dois parâmetros tal como por exemplo *verifyText //div//a[2] Login.* Dependendo do comando, os parâmetros poderão ser opcionais, aliás, alguns comandos não necessitam de parâmetros para serem executados. Segue-se um exemplo da utilização de parâmetros e comandos num *script* Selenium:

```
goBackAndWait
verifyTextPresent                          Welcome to My Home Page
type                    id=phone           (258) 824267-980
type                    id=address1        ${myVariableAddress}
```

Os parâmetros variam muito mas são, tipicamente, a localização que permite identificar o elemento do Interface de Utilizador na página, o texto para verificação do conteúdo esperado e/ou variáveis que podem referir-se a um campo de texto ou então uma opção de uma lista. Para que um *script* seja executado através do Selenium-IDE deve antes ser gravado num ficheiro com conteúdo HTML. O código HTML consiste numa tabela composta por três colunas. A primeira coluna identifica o comando, a segunda a variável, e a última o valor desta. Os valores das últimas colunas são opcionais pois dependem do comando a ser executado, contudo deverão estar sempre presentes. Cada linha da tabela refere-se a um novo comando Selenium tal como ilustra o exemplo que segue:

```
<table>
<tr><td>open</td><td></td><td>/download/</td></tr>
<tr><td>assertTitle</td><td></td><td>Downloads</td></tr>
<tr><td>verifyText</td><td>//h2</td><td>Downloads</td></tr>
</table>
```

O Selenium é composto por três ferramentas principais. Cada uma delas desempenha uma função específica no processo de desenvolvimento de teste automatizado para Aplicações Web. As três componentes são: i) Selenium-IDE é uma ambiente integrado de desenvolvimento que permite criar os *scripts* de teste; opera como um *add-on* para o *Firefox* e providencia um Interface simples para criar *scripts* de teste e executá-los, ii) Selenium-RC permite aos testadores utilizarem uma linguagem de programação mais flexível e extensível para desenvolver *scripts* de teste lógicos, iii) Selenium-Grid, possibilita que as soluções Selenium-RC estendam os seus *scripts* de teste para múltiplos ambientes.

4. PROPOSTA DO MODELO

4.1 Análise da Ferramenta de Testes a Utilizar

A escolha de ferramentas foi antecedida da escolha e avaliação dos critérios listados na Tabela 1. O passo seguinte foi a realização de experiências sobre a ISUPAC3 com cada uma das ferramentas e respectiva avaliação com base nos critérios previamente seleccionados. Os resultados apurados na Tabela 1 demonstram que a ferramenta que melhor se adaptou ao nosso cenário foi o *Sahi* daí a razão da sua escolha. Apresentamos em seguida a explicação dos critérios utilizados que serão valorados de maneira diferente conforme indicado na Tabela 1:

4.1.1 Compatibilidade e Browsers Suportados

A ferramenta de testes terá de ser compatível com a tecnologia AJAX utilizada na implementação da Interface do Utilizador da ISUPAC3. Para além disso, deverá funcionar, pelo menos, com os dois browsers suportados pela ISUPAC3, o *Microsoft Internet Explorer* e o *Mozilla Firefox*. Será considerada uma vantagem adicional o facto de os *scripts* de teste

gerados automaticamente pela ferramenta servirem para ambos os browsers pois desta forma simplificaria o processo de geração de testes, evitaria o esforço manual, bem como a possibilidade de o testador cometer erros.
Os Sistemas Operativos suportados pela ferramenta terão também de ser levados em conta. Terá de funcionar no Sistema Operativo *MS Windows Xp* e na distribuição do Linux Ubuntu 8.0.4 para Desktop. Por outro lado constituirá também uma vantagem o facto de a ferramenta permitir a integração com a ferramenta de gestão de tarefas denominada *Jira* actualmente utilizada pelos testadores da ISUPAC3 no registo de anomalias detectadas durante o processo de teste.

4.1.2 Linguagem de Programação dos Scripts

Os *scripts* de teste serão escritos numa certa linguagem de programação, manualmente ou através do método *capture/replay*. Por isso, um factor importante na escolha da ferramenta será a linguagem em que estes testes são escritos e a disponibilidade de documentação de apoio a esta mesma linguagem. O objectivo principal é que essa linguagem seja inteligível pois simplifica a leitura dos *scripts* gerados, a sua criação/alteração manual, a sua manutenção, entre outras vantagens.

4.1.3 Necessidade de Formação

É importante que a ferramenta escolhida não requeira muito tempo e recursos para o treino dos testadores sob pena de aumentar os custos de implementação do *Ambiente de Testes* do sistema automatizado. A simplicidade da ferramenta poderá levar os testadores a concentrarem os seus esforços no processo de produção de *scripts* de teste o que leva à obtenção de *scripts* de teste de maior qualidade. Com o crescimento da ISUPAC3 ao longo do tempo, antevemos a existência de s*cripts* de teste complexos, e dado que anualmente são recrutados novos testadores será uma vantagem ter uma ferramenta em que não careça de muita formação.

4.1.4 Estabilidade e Versões da Ferramenta de Testes

Um dos aspectos que deve ser levado em conta na escolha de um Software é a estabilidade do mesmo bem como o número de versões programadas ao longo do ano. É legitimo pensar que um Software que existe há bastante tempo e é actualizado com alguma periodicidade seja mais fidedigno. Provavelmente não fará muito sentido escolher uma ferramenta que tenha o seu desenvolvimento descontinuado ou que permanece sem alterações ao longo de um período de tempo prolongado. Não muito importante, mas que pode ser um indicador, tem que ver com a estabilidade da equipa que desenvolve o Software. Se esta se mantiver ao longo dos anos, é um bom indicador da estabilidade do mesmo, pois as mesmas pessoas trabalham sobre esse Software por muito tempo.

4.1.5 Interface Gráfica para Criação dos Casos de Teste

De forma a aumentar a produtividade no processo de criação de scripts de teste seria vantajoso se a ferramenta permitisse a criação de scripts de teste através de uma Interface gráfica fácil de ser utilizada. Se a ferramenta permitir uma forma expedita de criar scripts de teste combinada com a capacidade de poder melhorar os scripts alterando o código fonte, esta irá contribuir para a criação de *scripts* com maior qualidade.

4.1.6 Documentação, Suporte e Comunidade de Utilizadores

Uma ferramenta bem documentada, e com suporte permanente é uma mais-valia pois facilita o processo de aprendizagem do uso da mesma. Para além disso mostra-se muito útil no processo de instalação, configuração e manutenção de qualquer sistema informático. A quantidade de utilizadores que uma ferramenta tem a nível global também é um bom

indicador da maturidade e perfeição da mesma. Para além da documentação e da comunidade de utilizadores é também necessária uma garantia de suporte da ferramenta; nas ferramentas comerciais a garantia de suporte é sempre muito boa enquanto que no mundo das ferramentas abertas nem sempre há garantias de suporte.

4.1.7 Instalação e Configuração do Ambiente de Testes

Uma das etapas de implementação de um sistema de teste automático consiste na configuração do *Ambiente de Testes*. A instalação deste ambiente é mais complexa e relativamente mais cara quando comparada com o Ambiente de Testes manual. A complexidade da instalação e configuração constitui sem dúvida uma das variáveis para o cálculo do custo da implementação do Ambiente de Testes automatizado, daí a necessidade de avaliar esse custo para cada ferramenta de teste. Situação ideal seria poder-se instalar e configurar a ferramenta de teste em qualquer máquina cliente no menor tempo possível.

4.1.8 Limitações

Como é de esperar, qualquer ferramenta de automatização de testes apresenta as suas limitações, daí a importância de fazer uma análise cuidadosa entre as limitações apresentadas e o propósito da nossa aplicação de teste. Para isso foi necessário fazer uma análise do actual estágio da ISUPAC3 e tentar perspectivar o estágio a médio prazo por forma a permitir que a ferramenta escolhida não necessite de ser alterada no médio prazo por força da alteração estrutural da Aplicação.

4.1.9 Custos de Licenciamento

A ISUPAC3 é uma Aplicação maioritariamente desenvolvida com recurso a tecnologias de código aberto de licença livre. Será por isso uma vantagem encontrar ferramentas de automatização de testes com características semelhantes. Isto não significa que as ferramentas comerciais devam ser excluídas à partida, pois estas podem mostrar-se muito mais vantajosas nos outros critérios e apresentar opções de licenciamento de custos relativamente baixos. Uma atenção especial deve ser dada aos termos da licença da ferramenta por forma a garantir que esta não entra em conflito com a filosofia de desenvolvimento da ISUPAC3

NO	Critério	Pont.	Watir	Sahi	Badboy	Selenium
1	Compatibilidade e Browsers Suportados	10	4	10	2	8
2	Linguagem de Programação dos Scripts	10	10	10	2	10
3	Necessidade de Formação	10	10	10	8	10
4	Estabilidade e Versões da Ferramenta de Testes	10	10	10	8	8
5	Interface Gráfica para Criação dos Casos de Teste	5	0	5	3	5
6	Documentação, Suporte e Comunidade de Utilizadores	10	10	8	10	10
7	Instalação e Configuração do Ambiente de Testes	5	5	5	5	4
8	Limitações	5	2	4	1	4
9	Custos de Licenciamento	5	5	5	2	5
	Total	**70**	**56**	**67**	**41**	**64**

Tabela 1 - Matriz de escolha das ferramenta

4.2 Justificação da escolha da ferramenta Sahi

A ferramenta *Sahi* recorre à técnica *Capture/Replay* para criar e executar *scripts* de teste. Esta técnica consiste em simular eventos de um utilizador, como por exemplo pressionar um *link*

ou um botão sobre o *browser*, gravar a sequência da simulação e, posteriormente, reproduzir a sequência de execução gravada. Possui um editor gráfico de scripts de teste e permite a alteração do código gerado de forma simplificada o que possibilita melhorar a qualidade dos *scripts* gerados através da Interface gráfica.

A Figura 3 mostra a arquitectura da ferramenta *Sahi*. Contém um *browser* configurado para fazer pedidos através do *Proxy Sahi* que reencaminha os pedidos para o Servidor da Aplicação e a resposta segue o caminho inverso. A possibilidade dos pedidos e respostas *HTTP* serem feitas através do *Sahi* permite que este possa gravar e reproduzir todas as interacções entre o cliente (*browser*) e o Servidor de Aplicação.

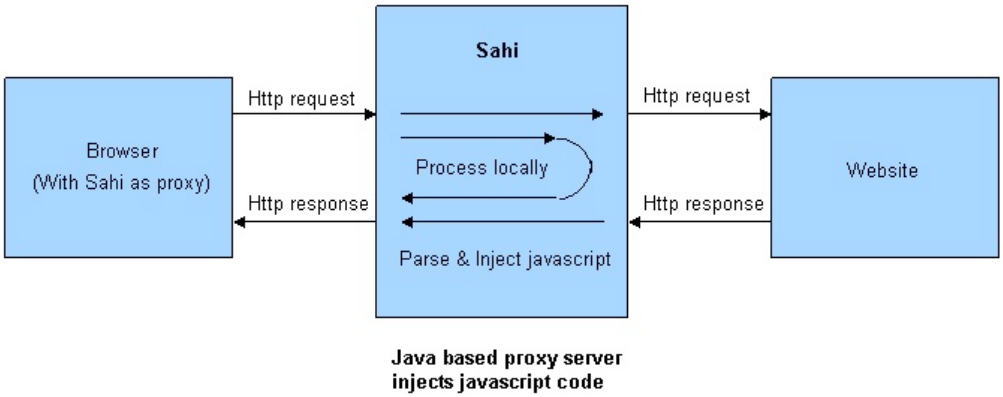

Figura 3 - Arquitectura da ferramenta *Sahi*
Fonte: [Peddi, 2008]

O *Sahi* utiliza o *Javascript* para simular os eventos o que é especialmente indicado para este propósito por ser suportado pela maior parte dos *browsers* e permite ainda o acesso aos vários elementos no *browser*. Os processos no *browser* são controlados através da injecção do *JavaScript* necessário para simular a Aplicação a partir de um Servidor *Proxy*.

4.3 Arquitectura proposta para o sistema de testes

A Figura 4 ilustra o sistema proposto para automatização dos testes da ISUPAC3. Consiste numa máquina cliente onde estão instalados pelo menos os browsers *Mozilla Firefox* e *MS Internet Explorer* e um *Deamon Sahi* que entre várias funções actuará como *Proxy* para aceder à ISUPAC3 instalada num servidor distinto.

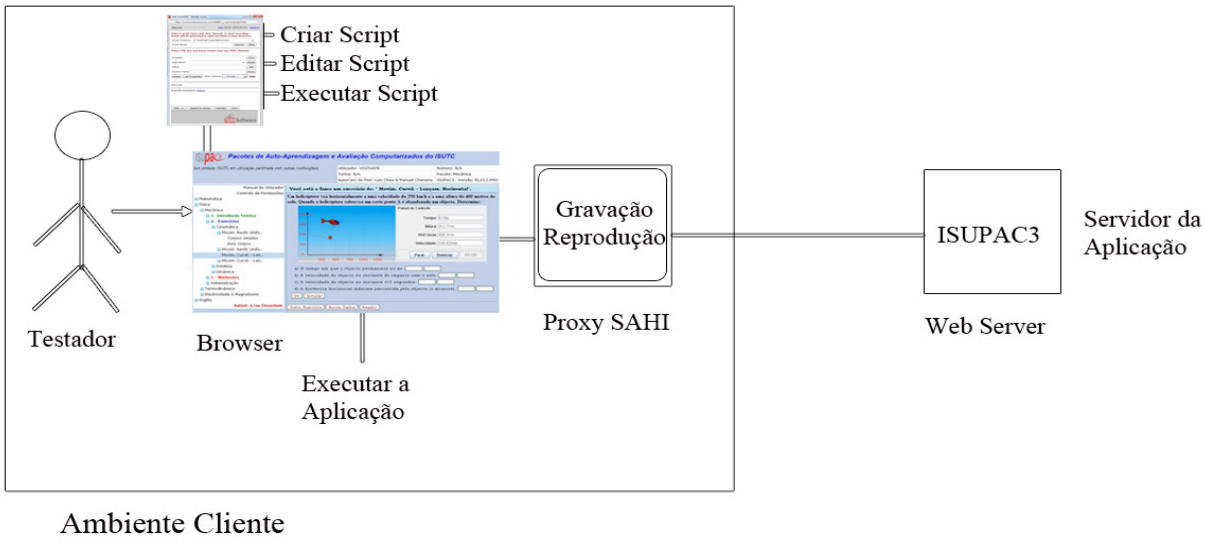

Figura 4 - Arquitectura de teste proposta para a ISUPAC3

Com o *Deamon Sahi* os testadores poderão construir os scripts de teste através do método *Capture/Replay*, com a possibilidade de melhorar os scripts manualmente. O mesmo *Deamon* permite ainda executar os scripts de teste. No fim de cada execução o *Deamon* emite um relatório de sucesso ou falha de cada script executado.

5. Impactos da Automatização da Execução de Testes na Qualidade da ISUPAC3

Como foi referido anteriormente, o objectivo deste trabalho é automatizar os casos de teste que actualmente são executados de forma manual sobre a ISUPAC3. Depois de analisar os casos de teste existentes, apesar de estes servirem para detectar um conjunto não desprezível de falhas da ISUPAC3 em desenvolvimento, constatámos que necessitam de ser melhorados em alguns aspectos nomeadamente, (i) existem casos de teste que testam várias funcionalidades em simultâneo, (ii) não estão previstos testes da não conformidade na maior parte dos casos, e (iii) os testes apresentam erros ortográficos o que dificulta o processo de leitura.

Em face dos problemas encontrados e da necessidade de formalização dos casos de teste numa linguagem de programação foram efectuadas as seguintes melhorias:

Modularização: Procedemos a esta simplificação pois a extensão da descrição dos casos de teste anteriores resultava não só numa complexidade exagerada de cada caso de teste mas também numa impossibilidade de verificação efectiva de qual a falha existente. Verificámos que cada caso de teste testava mais do que uma funcionalidade o que fazia com que a análise da real falha fosse muito difícil. Assim, durante o processo de codificação dos casos de teste procedemos à análise dos testes e decomposição dos mesmos em *componentes elementares de teste* de modo a que cada uma das componentes testasse apenas uma funcionalidade. Este processo permitiu-nos uma depuração dos casos de teste existentes, facilidade na manutenção dos casos de teste, e uma mais fácil análise das falhas resultantes.

Parametrização: Associada à questão da modularização verificámos que as componentes elementares de teste encontradas eram reutilizadas várias vezes. Dada a especificidade do *Sahi* e a possibilidade de definição de procedimentos de teste com recurso a parâmetros, a actividade de Modularização permitiu-nos ainda a reutilização de partes do código para a execução de testes.

Análise dos fluxos principais e alternativos de teste: ao contrário dos casos de teste manuais, que apenas consideravam os casos em que o sistema estava em conformidade com os requisitos, o processo de automatização levou-nos também a estudar a existência de fluxos alternativos que não devem ser permitidos. Sendo verdade que esta tarefa é ortogonal ao processo de automatização, pensamos que esta análise nunca teria sido feita se não tivéssemos iniciado esta actividade de automatização e consequente refactorização dos casos de teste.

Os casos de teste foram escritos de acordo com o padrão IEEE 829-1998 e foi utilizada a linguagem de *scripting* do Sahi para efeitos de automatização da execução destes. Foram seleccionados vários casos de teste para automatizar entre os quais a autenticação do utilizador, a expansão dos menus da aplicação, a consulta de textos pedagógicos, a resolução de exercícios, e a resolução de Minitestes. Neste artigo ilustramos o caso da autenticação do utilizador perante a aplicação.

5.1 Um exemplo de automação - Caso do teste de autenticação

a) **ID 0001:** Teste de autenticação

b) **Descrição do caso de uso:**

O caso de uso que se pretende testar é o acesso a certos recursos da ISUPAC3 tais como preparação de textos pedagógicos, exercícios, Minitestes, consulta de pautas, etc., que necessitam de prévia autenticação.

c) **Pré - requisitos:**

Par de autenticação *isupac3.professor/limeaa* e para o caso do aluno o par *isupac3.aluno/spel*.

d) **Pós - condições:**

A ISUPAC3 deverá solicitar autenticação quando o utilizador tentar aceder a recursos protegidos e esta deverá ser aceite para os casos em que o par utilizador/senha esteja correcto.

e) **Passos e resultados esperados:**

Passo	Executar	Resultado esperado
1	Digitar no browser o endereço: http://isupac3.isutc.transcom.co.mz	Aparecerá a página de rosto principal da ISUPAC3.
2	Seleccionar a sequência: *Matemática-Matrizes-Administração-Miniteste.*	Aparecerá uma caixa de autenticação.
2.1	Colocar o par *professor/senha-professor* e pressionar o botão *Entrar*.	No painel de execução aparecerá o ecrã dos minitestes.
2.2	No link *Autententicação e/ou Desautenticação* seleccionar a opção *Desautenticar-se.*	Será terminada a sessão do utilizador autenticado.
3	Seleccionar a sequência *Matemática-Matrizes-Administração-EdPac–Preparação de textos*	Aparecerá uma caixa de autenticação.
3.1	Colocar o par *professor/senha-professor* e pressionar o botão *Entrar*.	No painel de execução aparecerá o ecrã de preparação de textos.
3.2	No link *Autententicação e/ou Desautenticação* seleccionar a opção *Desautenticar-se*	Será terminada a sessão do utilizador autenticado.
4	Seleccionar a sequência *Matemática-Matrizes-Administração-EdPac–Preparação de exercícios*	Aparecerá uma caixa de autenticação.
4.1	Colocar o par *professor/senha-professor* e pressionar o botão *Entrar*.	No painel de execução aparecerá o ecrã de preparação de exercícios.
4.2	No link *Autententicação e/ou Desautenticação* seleccionar a opção *Desautenticar-se*	Será terminada a sessão do utilizador autenticado.
5	Repetir os passos 1, 2, 3 e 4 com o par *aluno/senha-aluno.*	Em todos os casos será aceite a autenticação mas as entradas *Minitestes*, *EdPac –Preparação* de textos e *EdPac–Preparação de exercícios* serão desabilitadas.

6	No link *Autententicação e/ou Desautenticação* seleccionar a opção *Autenticar-se*	Aparecerá uma caixa de autenticação.
6.1	Colocar o par *professor/senha-aluno*.	Será negada à autenticação.

Tabela 2 - Caso de teste de Autenticação

Analisando os passos de teste entre 2 e 6 constatámos que todos fazem exactamente a mesma coisa, isto é, o resultado esperado consiste na expansão de uma sequência de menus e na introdução na caixa de autenticação de um par *utilizador/senha*. Definimos por isso esta sequência como uma componente elementar de teste e procedemos à sua codificação e reutilização ao longo do processo de testes. O *script* de teste para esta componente elementar consiste numa função que recebe como parâmetros a sequência de passos para chegar ao menu pretendido e o par utilizador/senha necessário.

```
function autenticate ($label1,$label2,$label3,label4,$userName,$password){
        _click(_div(label1);
        _click(_div(label1);
        _click(_div(label1);
        _click(_div(label1);
        _setValue(_textbox(0), userName);
        _setValue(_password(0), password);
        _click(_cell("Entrar"));
        _click(_div("Autenticação e/ou Desautenticação"));
        _click(_cell("Desautenticar-se-se[1]"));
}
```

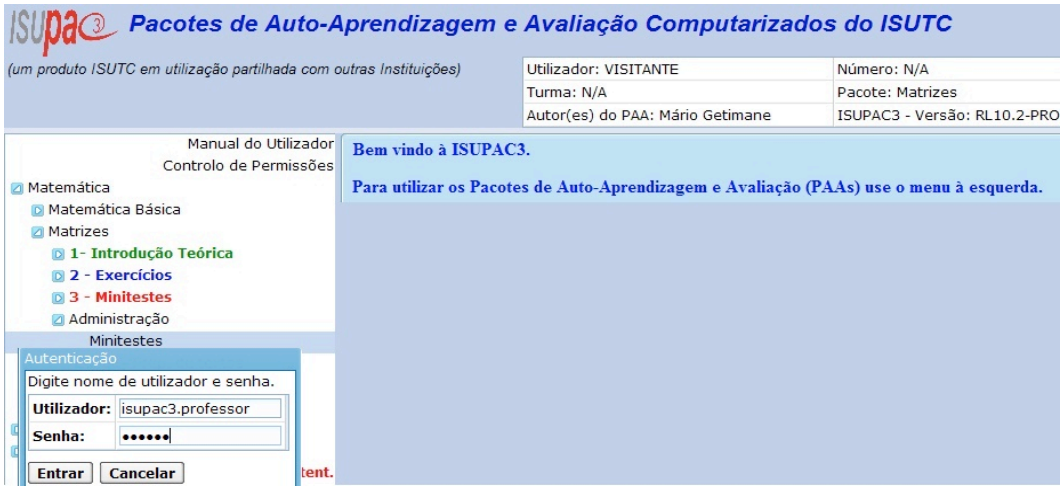

Figura 5 - Autenticação na ISUPAC3

Tomando o exemplo ilustrado na Figura 5, a função *autenticate* foi chamada com os parâmetros *Matemática, Matrizes, Administração, Minitestes, isupac3, professor/limeaa*. Esta Figura corresponde à execução dos passos 2-2.2 da Tabela 2.

Utilizando a função *autenticate* podem ser criados vários casos de teste, simples de serem percebidos e mantidos. Outra vantagem que se obtém através desta decomposição em componentes elementares de teste é a reutilização do código para várias situações similares. Os restantes passos de teste apresentados na Tabela 2, 3-6.1, podem ser realizados usando esta mesma função simplificando assim a sua implementação.

Procedendo a esta Modularização e Parametrização, o processo de manutenção dos testes será também simplificado pois o código dos testes é independente dos dados e estará armazenado

num local distinto pelo que permite uma maior flexibilidade na actualização de ambos, isto é, os *scripts* de teste podem ser actualizados independentemente dos dados e vice-versa.

Neste teste, no passo 6, exploramos já um pouco a existência de fluxos alternativos coisa que não acontecia nos testes realizados anteriormente. O estudo de fluxos alternativos, apesar de se ter iniciado com este processo de automatização dos testes, está ainda numa fase inicial e a sua concretização é por isso ainda trabalho futuro.

6. RESULTADOS OBTIDOS

6.1. Resultado da execução automática do teste (Tempo)

Para realizar a experiência foram configurados Ambientes de teste para dois testadores da CELTES (Célula de Testes da ISUPAC3) com a categoria de Analista 1 (categoria atribuída a testadores com experiência mínima de 2 anos, conhecedores da Aplicação). Os resultados obtidos da execução manual e automática dos casos de teste estão descritos na Tabela 3.

N	Descrição	TEM (min)	TEA (min)
1	Teste de autenticação do utilizador	5	1
2	Teste de expansão dos menus da aplicação	10	1
3	Teste de consulta de textos pedagógicos	5	1
4	Teste de resolução de exercícios	35	5
5	Teste de resolução de Miniteste	20	3

Tabela 3 - Resultados da execução dos testes

Legenda: TEM (min) – Tempo de Execução Manual em minutos

TEA (min) – Tempo de Execução Automática em minutos

Os resultados obtidos demonstram que executando os casos de teste de forma automática consome-se em média cerca de 10% do tempo necessário para executar os mesmos testes manualmente. Isto significa que automatizando a execução de testes passaremos a dispor de mais tempo para as outras etapas de teste como é o caso da análise dos resultados e a manutenção dos scripts de teste. Perspectivamos que a médio prazo os custos de teste da ISUPAC3 sofrerão uma redução significativa.

A redução de custos só será notada quando a CELTES atingir um razoável grau de automatização da execução dos testes. A julgar pelo tamanho da Aplicação perspectivamos que no curto prazo não seja necessário aumentar o actual efectivo de testadores que rondam os 10 elementos. Assim que for atingido o referido grau de maturidade na automatização poderemos reduzir para metade os recursos associados à execução de casos de teste.

6.2. Comparação entre a execução automática e manual de testes (Recursos)

Tal como se esperava no início deste projecto, a execução automática de testes face à sua execução manual apresenta grandes diferenças em termos dos recursos utilizados. As vantagens da execução manual estão relacionadas com a configuração do ambiente de testes que não é muito exigente e, uma vez criados os casos de teste é uma questão de executá-los manualmente. No entanto, muitas vezes a execução manual leva-nos a uma situação de não execução de todos os casos de teste por causa dos prazos que são impostos para execução dos mesmos.

Os resultados que obtivemos na execução automática dos testes foram bastante encorajadores o que nos reforça a ideia inicial de que o caminho da automatização é o mais acertado para melhorar a qualidade dos testes da ISUPAC3 e consequentemente melhorar a qualidade da Aplicação. Os tempos reduzidos, na ordem dos 10%, que obtivemos para a execução dos testes é um factor importante pois assim eventuais derrapagens de tempo no processo de desenvolvimento não terão tanto impacto como na situação da sua execução manual.

6.3. Impacto do processo de automatização na qualidade dos casos de teste

Verificámos que o projecto de automatização dos testes da ferramenta de e-learning ISUPAC3 actualmente em utilização no ISUTC, levou não só a uma melhoria da eficiência do processo de teste a nível de tempo, custos e recursos, como levou também a uma melhoria da qualidade do software produzido.

A análise do processo e casos de teste existentes levou-nos à detecção de alguns problemas, falhas, e imprecisões que conduziram à implementação dos processos de Modularização e Parametrização dos casos de teste, bem como ao estudo de fluxos de execução alternativos, conforme descrito na Secção 5.

6.4. Capacidade de detecção de novas falhas e/ou defeitos (Cobertura)

Os casos de teste da ISUPAC3 foram gerados com base nos casos de uso existentes. No processo de geração de casos de teste o fluxo de eventos constitui a parte mais importante pois é a partir destes que se obtêm informações sobre os fluxos básicos e os fluxos alternativos.

Constatámos que a maior parte dos casos de teste existentes, apenas testavam o funcionamento do fluxo básico do caso de uso, isto é, verificava-se apenas o comportamento esperado da Aplicação para os casos de sucesso. Isto fazia com que fossem apenas detectadas falhas referentes à execução normal de um caso de uso. Na utilização dos casos de uso, estes nem sempre são executados com sucesso. Por exemplo, para testar a autenticação de um utilizador, não é suficiente testar apenas a situação em que este se autentica com o par utilizador/senha correctos. É preciso verificar também o que acontece, por exemplo, se este tentar autenticar-se com o par utilizador/senha em branco, ou então com um par incorrecto, e assim por diante.

Por forma a melhorar a cobertura dos casos de teste da ISUPAC3 iniciou-se um processo de identificação e introdução nos casos de teste das situações de não conformidade. Na Tabela 4 apresentamos um exemplo referente ao caso de teste da Secção 5.1 contendo o fluxo básico e os fluxos alternativos.

ID	0001
Nome	Autenticação do Utilizador
Autor	Aluno, Técnico Pedagógico, Chefia
Descrição	O caso de uso permite que um utilizador tenha acesso à Aplicação mediante a introdução de um par *utilizador/senha* válido.
Prioridade	Muito alta
Frequência	90% dos utilizadores precisam de se autenticar para realizar diversas actividades, por exemplo, resolver Minitestes, elaborar Miniteste, entre outras
Pré-condições	O par *utilizador/senha* deve estar previamente registado na base de dados
Fluxo básico	1. O utilizador selecciona a opção de autenticar-se 2. Ser-lhe-á exibida uma caixa onde deverá digitar o par *utilizador/senha* registado 3. Ser-lhe-á permitido o acesso à Aplicação 4. A qualquer momento este poderá desautenticar-se
Alternativa 1	Utilizador tenta autenticar-se com o par *utilizador/senha-em-branco*. Deverá ser mostrada uma mensagem de erro e redireccionado para o ecrã de autenticação.
Alternativa 2	Utilizador tenta autenticar-se com o par *utilizador/senha-inexistente*. Deverá ser mostrada uma mensagem de erro e redireccionado para o ecrã de autenticação.
Alternativa 3	Utilizador tenta autenticar-se com o par *utilizador-nao-registado/senha-existente*.. Deverá ser mostrada uma mensagem de erro e redireccionado para o ecrã de autenticação.
Alternativa 4	Utilizador tenta autenticar-se mais de 3 vezes com pares *utilizador/senha* incorrectos. Deverá ser bloqueada a tentativa de autenticação e exibida uma mensagem para que este contacte o Administrador do Sistema.

Tabela 4– Análise dos fluxos da autenticação

Com base na especificação do caso de uso descrito na Tabela 4, nos fluxos básico e alternativos, é possível gerar especificações de testes com capacidade de detectar novos tipos de defeitos e/ou falhas aumentando assim a cobertura do nosso processo de testes. Este exemplo será seguido pelas demais especificações de teste actualmente existentes, passando assim a ISUPAC3 a ser sujeita também a testes de não conformidade.

7. CONCLUSÕES E TRABALHOS FUTUROS

7.1. Conclusões

Tomando como base as experiências realizadas durante a automatização dos testes da ISUPAC3 concluímos que com a automatização do processo de testes existiu uma redução significativa dos custos de execução. Nas experiências realizadas, a redução em termos de tempo de execução foi na ordem dos 90%, isto é, para executar o mesmo teste de forma automática precisámos de apenas 10% do tempo que seria necessário para executar a tarefa manualmente. Apesar desta redução significativa uma parte do tempo poupado teve de ser gasto no processo de análise dos resultados e na manutenção dos casos de teste no princípio do projecto de automatização. No entanto, esta situação será alterada quando o projecto atingir uma maior maturidade pois passará a ser preciso menos tempo para análise dos resultados, criação dos scripts de teste, bem como a manutenção dos casos de teste e assim os ganhos temporais serão bastante significativos.

Em relação ao tipos de falhas e/ou defeitos detectados pela automatização da execução dos casos de teste da ISUPAC3, todas as falhas que anteriormente eram detectadas continuaram a sê-lo visto que o trabalho consistiu fundamentalmente na automatização das actuais especificações de teste. No entanto, o exercício realizado levou-nos a aumentar a cobertura dos testes através da introdução de testes de não conformidade que antes não eram contemplados, bem como à simplificação dos casos de teste através da divisão de cada teste em unidades elementares permitindo-nos assim um melhor entendimento das funcionalidades a testar e uma simplificação do código de teste associado.

Concluímos assim que a automatização do processo de testes da ISUPAC3 contribuiu de três formas para a melhoria da aplicação: (1) a redução significativa em termos de tempo de execução permite que os testes sejam todos executados e não haja necessidade de descartar alguns por falta de tempo para os executar; (2) a avaliação dos testes existentes levou à simplificação de muitos em unidades elementares de teste que são mais fáceis de manter e simples de executar; e (3) a análise dos casos de teste levou-nos a constatar que a cobertura destes era deficiente nomeadamente em termos do teste de fluxos alternativos o que nos levou a estudar estes fluxos alternativos e consequentemente à melhoria da aplicação em análise.

7.2. Recomendações para trabalhos futuros

Durante a realização deste trabalho, enfrentámos várias dificuldades que nos indicaram alguns caminhos para trabalhos futuros bem como formas de tornar a CELTES mais eficiente na actividade de teste da ISUPAC3, e de outras Aplicações similares.

Uma das áreas de interesse tem que ver com a melhoria dos casos de teste da ISUPAC3. Apurámos que a maior parte dos casos de teste apresentavam algumas deficiências. Seria interessante encontrar um mecanismo que rescreva de uma forma uniforme tais especificações por forma a aumentar a cobertura de testes e consequentemente a detecção de falhas e/ou defeitos que possam surgir.

A outra área de interesse identificada tem que ver com a geração automática de casos de teste. A produção manual de casos de teste torna-se complicada para Aplicações com um número elevado de funcionalidades como é o caso da ISUPAC3 e o sucesso da automatização da

execução do processo de teste passa necessariamente pela existência de casos de teste capazes de detectar os diferentes tipos de defeitos e/ou falhas.

REFERENCIAS

Beck, K. Extreme Programming Explained. Addison-Wesley, 2000.

Bittencourt G. Inteligência Artificial Ferramentas e teorias. Florianópolis: Editora da UFSC, 2006.

Dougherty, A. K. Test Automation: Reducing Time to Market. International Conference on Software Testing, Analiysis & Review, 2002.

Fewster M. A. D Software Test Automation. Addison-Wesley 1999.

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

Oliveira, F. e Smith, C O desenvolvimento de Pacotes de Auto-Aprendizagem e Avaliação no ISUTC, 5º Congresso Luso-Moçambicano de Engenharia (CLME'08), Maputo, Moçambique, 2008.

Peddi N. Sahi – Web Automation and Test Tool. India: ThoughtWorks Ltd, 2008.

Pezze M.Y Teste e análise de software: Processos, Princípios e Técnicas. Porto Alegre: Bookman 2008

Pfleeger S Software Engineering: Theory and Practice. Prentice Hall 2001.

Sixpence, E. Projecto de Implementação de Testes Automatizados para a Aplicacão Web ISUPAC3 no ISUTC. Tese de Mestrado. ISUTC. Disponível em http://www.math.ist.utl.pt/~padao/teaching/students/2010-elton.pdf, 2010.

Sommerville A. Software Engineering. Addison-Wesley 2000.