ADAPTANDO TÉCNICAS DE TESTE DE SOFTWARE TRADICIONAIS PARA APLICAÇÕES WEB

ADAPTING TRADITIONAL SOFTWARE TESTING TECHNIQUES FOR WEB APPLICATIONS

Eder Carlos Salazar Sotto – eder.sotto@fatec.sp.gov.br
Gleydson Lucínio – gleydsonlucinio@gmail.com
Faculdade de Tecnologia de Taquaritinga (FATEC) –SP –Brasil

RESUMO

Este artigo apresenta a difícil tarefa de teste de aplicações *Web*, devido às peculiaridades de tais aplicações. O documento vai explorar as principais diferenças entre as aplicações *Web* e as aplicações *desktop* tradicionais, considerando as características de uma aplicação heterogênea composta de várias tecnologias e como essas diferenças afetam a atividade de teste.

Palavras-chave: Teste de software. Aplicações *Web*. Qualidade de *Software*.

ABSTRACT

This paper will present the difficult task of testing web applications, due to their peculiarities. It will explore the main differences between web applications and the traditional desktop ones, considering the characteristics of a heterogeneous application composed of several technologies and how these differences impact the testing activity.

Keywords: Software Testing. Web Applications. Software Quality.

1 INTRODUÇÃO

As primeiras aplicações *Web* eram estáticas e simplesmente estruturavam seu conteúdo em arquivos *Hypertext Markup Language* (HTML). A arquitetura utilizada nestas aplicações era cliente-servidor, onde a camada do cliente era composta por um navegador *Web*, capaz de interpretar e renderizar as páginas na tela. Já a camada do servidor é simplesmente encarregada de hospedar e fornecer os arquivos solicitados, necessários para o carregamento da página selecionada.

As aplicações *Web* vem sendo utilizadas em áreas críticas nas empresas, fazendo com que a ocorrência de falhas possa gerar enormes prejuízos. Por este motivo, a garantia da qualidade de *softwar*e é fundamental, e a demanda por ferramentas e metodologias para possibilitar o teste em aplicações *Web* acompanha seu crescimento (OFFUTT, 2004).

A arquitetura e as tecnologias utilizadas em uma aplicação *Web* mudaram drasticamente nos últimos anos. Se anteriormente as aplicações *Web* eram divididas em apenas duas camadas (cliente-servidor), hoje podem ser divididas em várias camadas, o que faz com que o teste destas aplicações também necessite de novas adequações, considerando que as aplicações *Web* atualmente são dinâmicas e heterogêneas (DELAMARO et al., 2007).

Uma aplicação *Web* heterogênea, segundo Conallen (2002) utiliza diversos componentes:

- HTTP (*Hypertext Transfer Protocol*) - protocolo utilizado para especificar como um navegador deve formatar e enviar uma solicitação para um servidor *Web*.
- HTML (*Hypertext Markup Language*) - linguagem para expressar a formatação visual de uma página ou documento. A HTML possui algumas marcas ou tags que informam ao navegador como deve mostrar a página *Web*. A tag anchor, por exemplo, é utilizada para criar ligações para uma página *Web*. A tag form permite a utilização de formulários que aceitam entradas dos usuários. Um formulário geralmente possui um botão para a sua submissão. A tag frame é utilizada para dividir a área de exibição do navegador em regiões ou quadros que apresentam o seu próprio arquivo HTML.
- *Scripts* - permitem a execução de comandos dentro de um documento HTML. Pode-se utilizar, por exemplo, JavaScript para escrever esses comandos.
- XML (*extensible Markup Language*) - linguagem de marcação para conteúdo, composta por um conjunto de elementos e atributos. Os elementos podem ser aninhados hierarquicamente. XML usa um esquema ou gramática para definir elementos, atributos, e todas as regras referentes aos dados que devem ser seguidas pelo documento XML. Os esquemas mais usados são: DTD (*Document Type Definition*) e XML *Schema* (XML *Language Schema*). Um documento XML é dito bem-formatado se ele obedece a determinadas regras sintáticas, como, por exemplo: a utilização de marcas de início (<marca>) e de fim (</marca>) para identificar o conteúdo dos elementos. Um documento XML bem-formatado e válido se ele segue as regras definidas por um esquema. Documentos XML podem ser passados como mensagens entre aplicações ou componentes *Web* para troca

de informações, podem ser usados para armazenar dados e para disponibilizar dados recuperados de uma base de dados relacional.

- *Web Services* - ou serviços *Web* são uma coleção de funções ou serviços disponíveis em uma rede, a serem utilizados pelos clientes. Para que isso seja possível, é necessário o envio de um documento que especifique o tipo de documento desejado. O SOAP (*Simple Object Access Protocol*) fornece um formato para esse tipo de documento. A linguagem WSDL (*Web Service Description Language*) é utilizada para descrever o serviço, e o UDDI (*Universal Description Discovery*) é um mecanismo utilizado para divulgar os serviços disponíveis para seus potenciais clientes.

Uma aplicação *Web* pode ser considerada como sendo um sistema distribuído, devido a sua arquitetura básica ser cliente-servidor, incluindo as principais características a seguir (LUCCA; FASOLINO, 2006):

- Um grande número de usuários distribuídos por todo o mundo acessando o sistema concorrentemente;
- Execução heterogênea em ambientes compostos por diferentes hardwares, conexões de rede, sistemas operacionais e navegadores;
- Uma gama extremamente heterogênea de componentes. Estes componentes podem ser construídos através de diferentes tecnologias e linguagens de programação;
- Capacidade de gerar componentes de *software* em tempo de execução de acordo com os dados de entrada fornecidos pelo usuário.

Cada aspecto descrito acima produz novos cenários de teste a serem considerados (LUCCA; FASOLINO, 2006).

Por serem dinâmicas, as aplicações *Web* acabam gerando desafios para o desenvolvimento e atividade de teste, conforme verificado a seguir (WU; OFFUTT, 2004):

- A equipe de desenvolvimento deve ser composta por pessoal com conhecimentos em diversas tecnologias;
- As tecnologias estão sempre evoluindo e se modificando, o que demanda modificações constantes nas ferramentas e métodos utilizados;
- Os requisitos da aplicação poderão ser alterados rapidamente, e por isso o tempo gasto em desenvolvimento e manutenção deve sempre ser o menor possível;
- Os componentes de *hardware* e *software*, embora sejam heterogêneos, devem ser integrados dinamicamente. Este critério de compatibilidade e interoperabilidade, embora

pouco críticos em aplicações tradicionais, são fundamentais quando se trata de aplicações *Web*;

- Neste tipo de aplicação, o controle da execução é diretamente afetado pelo usuário. Ao clicar em um *hiperlink* ou um botão, pode modificar totalmente o contexto de execução do programa;
- Mudanças na camada do cliente e do servidor ocorrem frequentemente. A aplicação e os dados no lado do cliente poderão sofrer alterações dinamicamente.

Embora os testes de aplicações *Web* compartilhem os mesmos objetivos dos testes de uma aplicação *desktop*, as teorias e métodos de testes tradicionais não podem ser utilizados simplesmente como são, devido a peculiaridades de uma aplicação *Web*, e precisam ser adaptados para esta nova abordagem.

2 TESTANDO OS REQUISITOS NÃO FUNCIONAIS DE UMA APLICAÇÃO *WEB*

Existem diferentes requisitos não-funcionais que uma aplicação *Web*, de forma explícita ou implícita, precisa satisfazer. Os principais requisitos incluem performance, escalabilidade, compatibilidade, acessibilidade, usabilidade e segurança. A seguir serão apresentados os sete principais requisitos não-funcionais de uma aplicação *Web* (LUCCA; FASOLINO, 2006).

1. **Teste de performance** - Tem o objetivo de verificar a performance do sistema, definido pelo tempo de resposta a cada solicitação. É executada simulando centenas ou mais de acessos simultâneos em um intervalo de tempo definido. As informações sobre os acessos são gravadas e então analisadas para estimar o nível de carga exaustivo dos recursos do sistema. Falhas descobertas no teste de performance são principalmente devido a falhas no ambiente de execução, como escassez de recursos ou recursos mal distribuídos;
2. **Teste de carga** - Permite que a performance do sistema seja avaliada com um nível pré-definido de carga. Tem como objetivo medir o tempo necessário para realizar várias tarefas e funções, utilizando o nível máximo de atividade suportado pela aplicação. Assim como no teste de performance, são simulados muitos acessos simultâneos. As informações são gravadas, e quando o teste não é executado dentro do tempo limite determinado, é gerado relatório de falhas. As falhas encontradas no teste de carga são principalmente devido a falhas no ambiente de execução;
3. **Teste de *stress*** - É utilizado para avaliar o sistema, ou componente, ultrapassando os limites especificados nos requisitos. Isso é utilizado para avaliar a resposta

do sistema em picos de atividade que possam exceder as limitações do sistema, e verificar se este está preparado para se recuperar destas condições. O teste de stress se difere do teste de performance e teste de carga, pois o sistema é executado além de seus limites, enquanto o teste de performance e carga simulam atividade regular dos usuários;

4. **Teste de compatibilidade** - Tem por finalidade revelar falhas durante o uso de diferentes plataformas de servidores *Web* ou navegadores, ou diferentes versões e configurações de ambos. A grande variedade de combinações possíveis de todos os componentes envolvidos na execução de uma aplicação *Web* não torna factível o teste de todas as possibilidades, mas a realização de algumas combinações pode ser considerado. Tanto a aplicação quanto o ambiente de execução são responsáveis pelas falhas de compatibilidade;

5. **Teste de usabilidade** - Tem por objetivo verificar em que nível a aplicação é fácil de utilizar. O teste de usabilidade é principalmente centrado no teste da interface do usuário. Trata de questões relativas a renderização correta do conteúdo, como gráficos e texto, além da clareza das mensagens, alertas e comandos são considerados e verificados. A usabilidade é um assunto crítico para uma aplicação *Web*, pois de fato pode determinar o sucesso ou fracasso da aplicação;

6. **Teste de acessibilidade** - Pode ser considerado como um tipo específico de teste de usabilidade, pois tem como objetivo verificar se o acesso ao conteúdo da aplicação é permitido até na presença de configurações reduzidas de *hardware* e *software* do lado do cliente, bem como o acesso por usuários com capacidade física reduzida (como pessoas com deficiência visual);

7. **Teste de segurança** - O objetivo deste tipo de teste e verificar a efetividade da segurança em todo o sistema *Web* contra acessos indesejados de usuários não autorizados, bem como sua capacidade de preservar os recursos do sistema de usuários impróprios, assim como garantir o acesso para usuários autorizados a seus respectivos serviços e recursos. Vulnerabilidades do sistema afetando a segurança podem estar contidos no código da aplicação, ou em qualquer dos componentes utilizados no sistema. Por possuir implementações heterogêneas, além de grande quantidade de usuários simultâneos oriundos de qualquer lugar, as aplicações *Web* acabam sendo mais vulneráveis que as aplicações tradicionais.

3 MODELOS DE REPRESENTAÇÃO DE APLICAÇÕES *WEB*

Em testes de *software*, são necessários modelos para representar os conceitos e relacionamentos sobre os itens a testar (BINDER, 2000): estes modelos podem ser utilizados como apoio a seleção de casos de teste efetivos, expressando comportamento obrigatório ou concentrando em aspectos de estrutura de *software* onde se suspeita da existência de erros.

Os modelos de aplicações *Web* foram fornecidos por várias metodologias para desenvolvimento de aplicações *Web*. Estes modelos utilizam modelos de *software* tradicionais, adicionando novos recursos para representar características específicas de aplicações *Web* (LUCCA et al., 2002).

Conallen (2002), propõe extensões de diagramas UML com novos estereótipos para representar componentes específicos de aplicações *Web*, tais como páginas HTML, formulários, páginas de servidor, etc.

Além destes modelos, mais modelos de representação são propostos para serem usados no teste de aplicações *Web*, como os modelos de comportamento. Estes modelos descrevem a funcionalidade destas aplicações independente de sua implementação, ou de modelos estruturais, derivados da implementação da aplicação. Enquanto modelos de comportamento suportam o teste da caixa-preta, modelos estruturais podem ser testados apenas com execução de teste de caixa-branca. Os modelos de comportamento utilizam casos de uso, tabela de decisões e máquina de estados, utilizando as técnicas da caixa-preta. A categoria de modelos estruturais pode utilizar modelos de controle de fluxo para os componentes da aplicação *Web*, e modelos descrevendo a organização da aplicação em termos de páginas e *links* entre elas (LUCCA; FASOLINO, 2006).

Lucca, Fasolino e Tramontana (2004) também propõem a criação de meta-modelos de uma aplicação *Web*. Este meta- modelo é apresentado na Figura 1. Trata-se de um diagrama de classes onde vários tipos de classes e associações representam várias categorias de componentes de uma aplicação *Web*, além de relacionamento entre elas. Uma aplicação *Web* pode ser modelada por um diagrama de classes UML instanciado a partir deste meta-modelo.

O meta-modelo considera que uma aplicação *Web* é formada por páginas *Web*, que podem ser divididas entre páginas de servidor, que são páginas processadas exclusivamente no servidor, e páginas do cliente, que são páginas retornadas pelo servidor para o cliente em resposta a uma solicitação. As páginas do cliente podem ainda ser divididas entre página estática, que é uma página armazenada no servidor, devolvida como resposta a uma solicitação sem que seu conteúdo seja modificado, e páginas dinâmicas, que são páginas

geradas dinamicamente em tempo real pelo servidor para envio ao cliente. A página do cliente é composta por código HTML, e pode conter imagens, *scripts* para execução no cliente como JavaScript, recursos multimídia como áudio e vídeo, entre outros. Uma página do servidor pode conter outras páginas do servidor e também associação com outros tipos de interface, permitindo conexão com um servidor de banco de dados, servidor de arquivos, servidor de e-mail, entre outros sistemas (LUCCA; FASOLINO; TRAMONTANA, 2004).

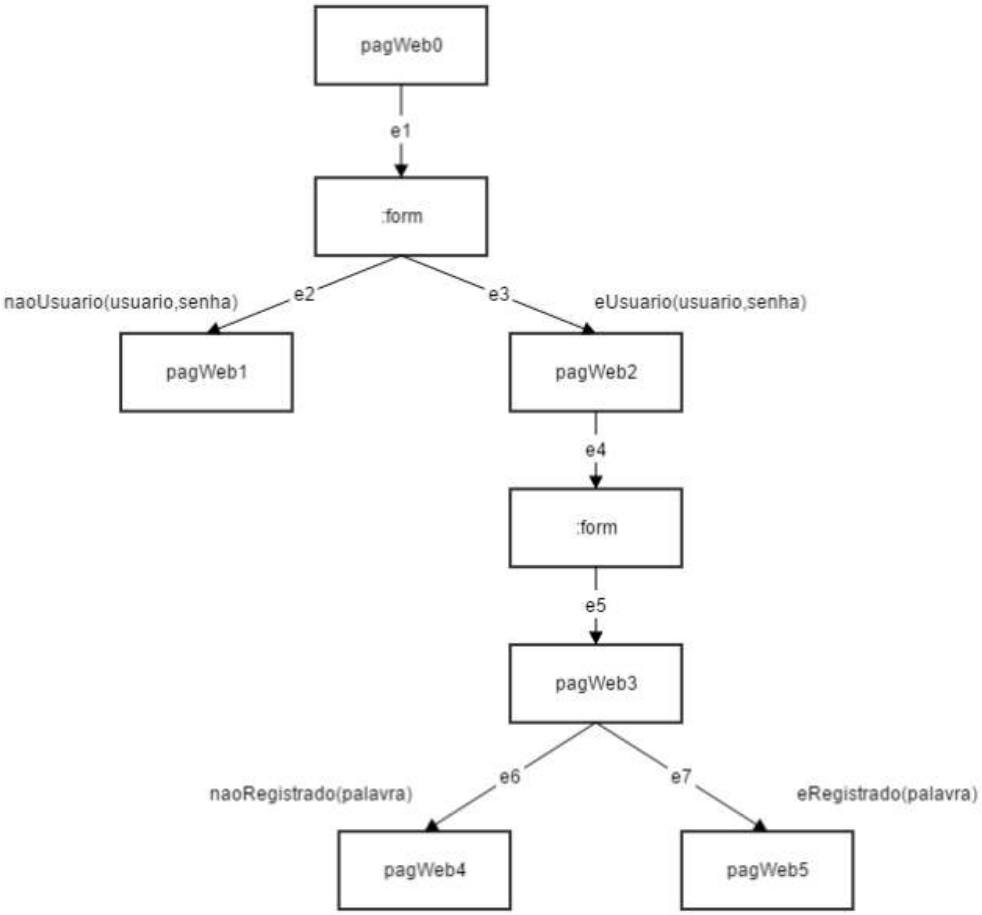
Critérios de teste estrutural aplicados a programas tradicionais são estendidos para o teste de aplicações *Web* e empregados no teste dinâmico. Ricca e Tonnela (2000), propõem os seguintes critérios:

- Teste de página (*page testing*) - cada página na aplicação deve ser visitada pelo menos uma vez em algum caso de teste;
- Teste de ligação (*hiperlink testing*) - cada ligação de cada página na aplicação deve ser percorrida pelo menos uma vez;
- Teste de Definição-Uso (*definition-use testing*) - todos os caminhos de navegação que exercita cada associação definição-uso devem ser exercitados;
- Teste de Todos-Usos (*all-uses testing*) - pelo menos um caminho de navegação que exercita cada associação definição-uso deve ser exercitado;
- Teste de Todos-Caminhos (*all-paths testing*) - cada caminho na aplicação deve ser percorrido pelo menos uma vez em algum caso de teste (RICCA; TONELLA, 2001).

Os critérios Teste de Definição-Uso e Teste de Todos-Caminhos nem sempre são aplicáveis, pois podem requerer um número excessivo de elementos.

Consideremos a aplicação *Web* de biblioteca, cujo fragmento do meta-modelo é apresentado na Figura 1, a qual indica os caminhos de navegação que podem ser percorridos na aplicação. A expressão de caminho de navegação e1(e2+(e3e4e5(e6+e7))), obtida na Figura 1, juntamente com os dados de entrada, disponibilizados pelo testador, é utilizada para gerar casos de teste para satisfazer o critério Teste de Página (DELAMARO et al., 2007).

Figura 1 – Meta-modelo de uma aplicação *Web* de biblioteca



Fonte: Delamaro et al. (2007, p. 220)

São apresentados abaixo três casos de teste para cobrir o critério Teste de Página, obtidos da instância do meta-modelo para a aplicação *Web* apresentada na Figura 1.

1. pagWebO, pagWeb1{usuario, senha}
2. pagWebO, pagWeb2{usuario, senha}, pagWeb3, pagWeb4{palavra}
3. pagWeb0, pagWeb2{usuario, senha}, pagWeb3, pagWeb5{palavra}

4 TESTE DE UNIDADE, INTEGRAÇÃO E SISTEMA DE UMA APLICAÇÃO *WEB*

O processo de teste de *software* define um conjunto de etapas de atividades de teste, onde em cada etapa considera diferentes níveis de teste. Geralmente, o teste de unidade é a primeira atividade do processo, seguido pelo teste de integração e de sistema. O teste de unidade é utilizado para verificar cada código-fonte de componente da aplicação individualmente, enquanto o teste de integração combina partes desta aplicação para verificar

como elas trabalham juntas. Por fim, o teste de sistema tem por objetivo descobrir defeitos que são relativos ao sistema, ao invés de um componente isolado (LUCCA; FASOLINO, 2006).

4.1 Teste de Unidade

Diferentes tipos de unidade podem ser identificados em um modelo de aplicação *Web*, conforme demonstrado na Figura 1 como páginas *Web*, scripts, formulários, servlets, ou qualquer outro tipo de objeto que possa ser incluído nessas páginas. Cada página pode ser considerada página do cliente ou página do servidor.

4.1.1 Teste da página do cliente

A página do cliente é a camada de visualização para o usuário final, permite a navegação do usuário dentro da aplicação. O teste de páginas do cliente geradas dinamicamente é um caso específico de teste. O problema básico deste tipo de teste é a disponibilidade de criar páginas que dependam da capacidade de identificar e reproduzir as condições para cada pagina ser criada. Um segundo problema é a quantidade de combinações que podem existir na criação de páginas dinâmicas, considerando estados da aplicação e entradas fornecidas pelo usuário. O critério de partição em base de equivalência pode ser utilizado para reduzir esta quantidade de combinações. O teste de unidade pode utilizar técnicas de teste funcional (caixa-preta) ou estrutural (caixa-branca). A seguir serão apresentadas falhas típicas encontradas no teste da página do cliente (LUCCA; FASOLINO, 2006):

- Diferenças entre o conteúdo apresentado pela página e o especificado e esperado pelo usuário;
- Destino errado de *links* direcionando para outras páginas;
- *Links* inexistentes ou redirecionando para páginas inexistentes;
- Ação incorreta realizada quando um botão ou qualquer outro tipo de objeto e selecionado pelo usuário;
- Falhas em scripts inclusos na página do cliente.

4.1.2 Teste da página do servidor

As páginas do servidor têm como principal responsabilidade implementar as regras de negócio da aplicação, coordenando sua execução e gravando e recuperando informações de uma base de dados.

Geralmente, páginas do servidor são implementadas por um conjunto de tecnologias e pode conter HTML, linguagens de script, servlets, entre outros.

O teste de unidade de páginas do servidor pode utilizar técnicas do teste estrutural e do teste funcional, assim como no teste de páginas do cliente.

A seguir serão demonstradas falhas típicas encontradas em páginas do servidor (LUCCA; FASOLINO, 2006):

- Falha na execução do script;
- Problemas no acesso aos dados no banco de dados;
- Falhas devido a existência de links incorretos entre as páginas;
- Defeitos na página cliente gerada dinamicamente, onde não foi cumprida a saída esperada para a página do servidor.

4.2 Teste de integração

O teste de integração de uma aplicação *Web* analisa conjuntos de páginas *Web* relacionadas, a fim de avaliar como trabalham em conjunto, identificando falhas ocorridas em seu acoplamento. Estratégias de integração adequada e critérios deverão ser definidos. Apresentar estes relacionamentos entre as páginas na documentação pode ser uma estratégia a ser utilizada para alcançar este objetivo. Por padrão, o modelo da aplicação *Web* pode ser obtido através do meta-modelo apresentado na Figura 1, podendo ser utilizado para identificar as páginas e seus relacionamentos.

No nível de teste de integração, o conhecimento sobre estrutura e comportamento da aplicação *Web* é necessário e deverá considerar que a estrutura da aplicação deve ser utilizada para definir o conjunto de páginas a serem integradas, enquanto o comportamento definido para estas páginas será verificado para a realização do teste de integração, que poderá ser feito utilizando uma estratégia de teste funcional (LUCCA; FASOLINO, 2006).

4.3 Teste de sistema

O teste de sistema tem por objetivo descobrir defeitos que dependem de toda a aplicação *Web*. Os testes de caixa-preta são geralmente explorados para realizar o teste de sistema e identificar falhas no comportamento da aplicação.

O Critério de cobertura típico para teste de aplicações *Web*, segundo Lucca e Fasolino (2006), deve incluir:

- Cobertura das funções do usuário/casos de uso (utilização dos testes de caixa-preta);
- Cobertura das páginas do cliente e do servidor (utilização dos testes de caixa-branca);
- Cobertura dos *links* (utilização dos testes de caixa-branca);

5 TESTE BASEADO EM DEFEITOS

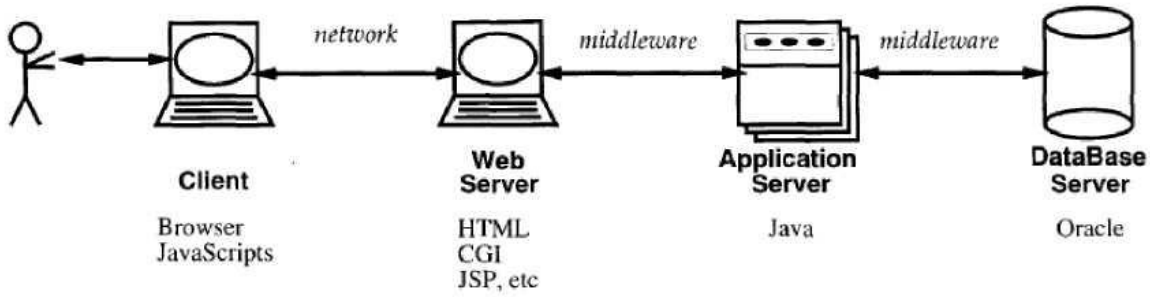
Atualmente as aplicações *Web* são compostas por uma arquitetura multicamadas, onde cada componente se comunica com componentes dispostos em camadas adjacentes.

Os critérios de teste baseados em defeitos podem ser utilizados para testar alguns aspectos de aplicações *Web*, como a interação entre componentes distintos através de mensagens XML.

5.1 Mutação de interação em documentos XML

A Figura 2 ilustra um modelo comum de uma aplicação *Web*. Ao invés de um modelo cliente-servidor simples, a configuração apresentada utiliza um modelo multicamadas. A camada do cliente é representada pelo navegador *Web*, utilizado pelo usuário para acessar a aplicação *Web*, onde cada aplicação é hospedada e fornecida por servidores *Web*. Para aumentar alguns fatores como segurança, confiabilidade, disponibilidade e escalabilidade, assim como funcionalidade, os componentes podem ser dispostos em servidores separados, onde cada servidor de aplicação funciona em paralelo. Uma das formas de comunicação entre componentes, seja dentro de uma mesma camada ou entre camadas, é através do XML (LEE; OFFUTT, 2001).

Figura 2 – Modelo de aplicação *Web* cliente-servidor com uso de *Web Services*



Fonte: Lee; Offutt (2001, p. 202)

Embora o XML seja uma forma conveniente de transmissão de dados, o número e a diversidade de componentes de *software* em uma aplicação *Web* moderna faz com que hajam grandes chances de existência de erros. Um problema que dificulta a correção destes erros, assegurando que estes componentes se comuniquem de forma correta, é a quantidade de componentes e a quantidade de interações de tipos distintos dentro de uma mesma aplicação. Cada tipo de interação corresponde a um padrão XML diferente, e este precisa passar por análise que permita verificar se está em formato correto, ou seja, obedecendo ao padrão estabelecido, verificando também erros de sintaxe e semântica (EMER et al., 2005).

Lee e Offut (2001) aplicam análise de mutantes para a validação da interação de dados por meio de mensagens XML entre componentes de uma aplicação *Web*. Este tipo de abordagem e chamada mutação de interação.

O teste de mutação de interação utiliza um modelo de especificação de interação (*Interection Specification Model* - ISM) para auxiliar na geração dos mutantes em mensagens XML. Esse modelo é composto da DTD (*Document Type Definition*), responsável por especificar a estrutura da mensagem XML de uma interação entre componentes *Web*, onde são definidas as mensagens XML de requisição e de resposta, além do conjunto de restrições (DELAMARO et al., 2007).

Após o estabelecimento do modelo de interação, é necessário criar operadores de mutação. Os operadores de mutação de interação (*Interation Mutation Operators* - IMOs) são regras aplicadas as mensagens XML para obtenção dos mutantes, onde são geradas mensagens XML modificadas. Como cada interação possui seu DTD específico, não é possível criar operadores de mutação que se apliquem em qualquer aplicação (EMER et al. 2005).

Inicialmente, duas classes de operadores de mutação foram criadas baseadas em restrições XML (EMER et al. 2005):

- **NOT *memrberOf*** - criada a partir da restrito *memberOf*, relacionada aos filhos de um elemento da mensagem XML;
- **NOT *lenOf*** - criada a partir da restrito *lenOf*, associada a quantidade de caracteres da *string* do conteúdo de um elemento na mensagem XML.

O processo de teste é realizado em uma interação I definida no modelo de interação por meio de um conjunto de casos de teste. Cada caso de teste é uma instancia mutante de uma interação na forma de mensagem XML, onde o operador de mutação foi aplicado (DELAMARO et al., 2007).

Uma aplicação *Web,* a qual busca informações sobre o acervo de uma biblioteca, precisa fornecer um nome de usuário e senha válidos para acesso ao sistema. Neste caso, a mutação de interação poderia ser aplicada na mensagem de requisição. A classe genérica do operador de mutação NOT *memberOf* poderia ser aplicada no elemento usuário. Já a classe genérica do operador de mutação NOT *lenOf* poderia ser aplicada no elemento senha, supondo que tenha sido especificado no ISM *lenOf* igual a seis (EMER et al. 2005).

A Figura 3 apresentada um exemplo de mensagem XML de requisição, contendo a mensagem original e duas mensagens mutantes, conforme os operadores de mutação NOT *memberOf* e NOT *lenOf*, além das respectivas mensagens XML de resposta (DELAMARO et al., 2007):

Figura 3 – Mensagens XML de requisição e resposta

Mensagem de requisição original
`<?xml version=1.0?>`
`<requisicao>`
`<usuario>Pedro</usuario>`
`<senha>Rs12aM</senha>`
`</requisicao>`
	Mensagem de resposta
	`<?xml version=1.0?>`
	`<resposta>`
	`<validacao>aceita</validacao>`
	`<usuario>Pedro</usuario>`
	`</resposta>`

Mensagem de requisição mutante aplicando o operador NOT memberOf
`<?xml version=1.0?>`
`<requisicao>`
`<usuario>Paulo</usuario>`
`<senha>Rs12aM</senha>`
`</requisicao>`
	Mensagem de resposta
	`<?xml version=1.0?>`
	`<resposta>`
	`<validacao>negada</validacao>`
	`<usuario>Paulo</usuario>`
	`</resposta>`

Mensagem de requisição mutante aplicando o operador NOT lenOf
`<?xml version=1.0?>`
`<requisicao>`
`<usuario>Pedro</usuario>`
`<senha>Rs12</senha>`
`</requisicao>`
	Mensagem de resposta
	`<?xml version=1.0?>`
	`<resposta>`
	`<validacao>negada</validacao>`
	`<usuario>Pedro</usuario>`
	`</resposta>`

Fonte: Delamaro et al. (2007, p. 222)

5.2 Perturbação de valores de dados

O modelo de interação de serviços *Web* considerado no teste é de dois a dois, ou seja, dois serviços *Web* se comunicando através de chamada de procedimento remota (*Remote Procedure Call* - RPC) ou um documento XML utilizando SOAP (DELAMARO et al., 2007).

A mutação de mensagem XML por perturbação de dados pode ser realizada de duas formas: perturbação por valor de dado e perturbação por interação.

A (*Data Value Perturbation* - DVP) é uma técnica de mutação que modifica os valores em mensagens SOAP de acordo com regras definidas nos tipos de valores, aplicando valores limites para tipos primitivos de dados como: *decimal, float, double, string e boolean* (OFFUTT; XU, 2004).

A perturbação de interação altera mensagens em RPC e em comunicação de dados.

Os operadores de mutação para uso de dados de tipo numérico segundo Delamaro et al. (2007) são:

- Divide(n) : altera o valor de n por 1/n, sendo n do tipo *double*;
- Multiply(n) : altera o valor de n por n x n;
- Negative(n) : altera o valor de n por -n;
- Absolute(n) : altera o valor de n pelo absoluto de n;

Os operadores de mutação para uso de SQL, segundo Delamaro et al. (2007) são:

- Exchange(n1,n2) : troca o valor de n1 e n2;
- Unauthorized(str): troca o valor de str por str "OR"1" = "1.

Considerando uma aplicação *Web* que busca por informações de um acervo de uma biblioteca, suponha que a validação de usuário e senha seja realizada por meio de mensagem SOAP, na qual os conteúdos dos elementos associados ao usuário e senha fornecidos sejam utilizados em uma consulta SQL. Em seguida é apresentado um exemplo de uso do operador *Unauthorized().*

O exemplo da Figura 4 apresenta o corpo das mensagens SOAP de requisição original e mutante, com uso do operador de mutação *Unauthorized()*, e uma consulta SQL utilizando o conteúdo modificado por esse operador de mutação (DELAMARO et al., 2007).

Figura 4 – Corpo de mensagem SOAP e consulta SQL

Corpo da mensagem SOAP de requisição original

```
<soapenv:Body>
<adminLogin soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <arg0 xsi:type="xsd:string">Pedro</arg0>
    <arg1 xsi:type="xsd:string">Rs12aM</arg1>
</adminLogin>
</soapenv:Body>
```

Corpo da mensagem SOAP de requisição mutante, após aplicação do operador de mutação Unauthorized()

```
<soapenv:Body>
<adminLogin soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <arg0 xsi:type="xsd:string">Pedro</arg0>
    <arg1 xsi:type="xsd:string">Rs12aM "OR" 1 "=" 1</arg1>
</adminLogin>
</soapenv:Body>
```

Consulta SQL com conteúdo modificado pelo operador de mutação Unauthorized()

```
SELECT usuário FROM usuarios
    WHERE usuário="Pedro" AND senha="Rs12aM" OR "1"="1"
```

Fonte: Delamaro et al. (2007, p. 224)

6 CONSIDERAÇÕES FINAIS

Neste trabalho foram relatados pontos comuns entre o teste de aplicações tradicionais e aplicações *Web*, apesentando as novas abordagens de conhecimento e entendimento obrigatórios para realização de qualquer tipo de teste bem-sucedido neste tipo de aplicação.

A principal conclusão é que todos os aspectos de teste são diretamente dependentes das tecnologias de implementação e da arquitetura utilizada, e qualquer abordagem de teste a ser realizado precisa ser profundamente adaptado à natureza heterogênea e dinâmica das aplicações *Web*.

Como tendências futuras na evolução das metodologias de teste para aplicações *Web* é possível que o novo desafio para o futuro seja a introdução de métodos ágeis em processos de teste para este tipo de aplicação, melhorando sua eficácia e eficiência, e seguindo uma tendência já verificada nas etapas de análise e desenvolvimento de *software*.

REFERÊNCIAS

BINDER, R. (2000). **Testing object-oriented systems: models, patterns, and tools.** Addison- Wesley Professional.

CONALLEN, J. (2002). **Building Web Applications with Uml.** Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.

DELAMARO, M.; MALDONADO, J.; JINO, M. (2007). **Introdução ao teste de software**. CAMPUS - RJ.

EMER, M. C. F.; VERGILIO, S. R.; JINO, M. (2005). **A testing approach for xml schemas.** In Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International, volume 2. IEEE.

LEE, S. C.; OFFUTT, J. (2001). **Generating test cases for xml-based web component interactions using mutation analysis.** In Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on. IEEE.

LUCCA, G. A. D.; FASOLINO, A. R. (2006). **Testing web-based applications: The state of the art and future trends.** Information and Software Technology. Quality Assurance and Testing of Web-Based Applications.

LUCCA, G. A. D.; FASOLINO, A. R.; FARALLI, F.; DE CARLINI, U. (2002). **Testing web applications.** In Software Maintenance, 2002. Proceedings. International Conference. IEEE.

LUCCA, G. A. D.; FASOLINO, A. R.; TRAMONTANA, P. (2004). **Reverse engineering web applications: the ware approach.** Journal of Software maintenance and evolution: Research and practice.

OFFUTT, J.; XU, W. (2004). **Generating test cases for web services using data perturbation.** ACM SIGSOFT Software Engineering.

RICCA, F.; TONELLA, P. (2001). **Analysis and testing of web applications.** In Proceedings of the 23rd international conference on Software engineering. IEEE Computer Society.

WU, Y.; OFFUTT, J. (2002). **Modeling and testing web-based applications.** GMU ISE Technical.