Testes e Geração de Código de Sistemas Web

Eduardo Aranha[*], Paulo Borba[†]

Centro de Informática – Universidade Federal de Pernambuco
Caixa Postal 7851 – 50732-970 Recife, PE
E-mail: {ehsa, phmb}@cin.ufpe.br

Resumo

Devido aos seus reconhecidos benefícios, a realização de testes vem sendo enfatizada em diversos processos e metodologias de desenvolvimento. Entretanto, por ser trabalhosa e entediante, a atividade de teste causa, em geral, um impacto negativo na produtividade a curto prazo. Este trabalho apresenta um processo de teste que utiliza geradores de código para melhorar a produtividade do desenvolvimento. Estes geradores são baseados na linguagem WSat que é introduzida por este trabalho e utilizada para realizar testes de sistemas Web como os de aceitação e de performance.

Palavras chaves: testes, sistemas Web, geração de código, processo de teste, test first design.

Abstract

Because of its recognized benefits, the testing activity has been emphasized by several development processes and methodologies. However, testing is, in general, laborious and tedious, causing, at first, a negative impact in development productivity. This work presents a test process that uses code generators to improve development productivity when testing is a major concern. These generators are based in the WSat language which is introduced in this paper and used to do Web systems tests like acceptance and performance tests.

Keywords: tests, Web systems, code generation, test process, test first design.

1 Introdução

Visando o desenvolvimento de sistemas corretos e robustos, a engenharia de software tradicionalmente enfatiza em suas metodologias de desenvolvimento a construção e automação de testes, como os testes de aceitação e de performance. Metodologias práticas mais recentes como o RUP [10] e *Extreme Programming* (XP) [4], tem destacado a atividade de teste como uma das práticas de programação chaves para o sucesso de sua implantação. Em XP, a construção destes testes é realizada antes mesmo da implementação do código testado (*Test First Design*) [6].

É esperado que a atividade de teste aumente a longo prazo a corretude e produtividade geral do desenvolvimento, dado que defeitos nos sistemas são descobertos mais cedo, diminuindo o

[*]Suportado parcialmente pelo IPAD.
[†]Suportado parcialmente pela CNPq, processo 521994/96-9.

trabalho de sua manutenção. Entretanto, por ser trabalhosa e entediante, a atividade de teste causa, em geral, um impacto negativo na produtividade a curto prazo. Consequentemente, atrasos no desenvolvimento do sistema acabam sendo naturalmente compensados através da redução ou eliminação da atividade de teste. Este fato implica, em geral, na diminuição da qualidade dos sistemas produzidos.

Visando suportar a programação de casos de teste de sistemas Web com um alto nível de abstração, definimos a linguagem de teste WSat (*Web System Acceptance Test language*) [2][3]. Para a utilização eficaz desta linguagem, definimos um processo de teste. Este processo visa também melhorar a produtividade com a realização dos testes em sistemas Web através do uso de dois geradores de código. Estes geradores, além de melhorar a produtividade, auxiliam a criação dos testes antes da implementação do próprio sistema (*Test First Design*).

Em primeiro lugar, introduzimos brevemente na Seção 2 a linguagem WSat, utilizada para realização de testes em sistemas Web. Em seguida, na Seção 3, apresentamos o processo de teste definido por este trabalho. Nas Seções 4 e 5, apresentamos geradores de código criados para suportar o processo de teste e melhorar a produtividade a curto prazo. Um experimento realizado é apresentado na Seção 6. Por fim, apresentamos as conclusões obtidas com a realização deste trabalho.

2 A Linguagem WSat

Para auxiliar o desenvolvimento de testes em sistemas Web, foi definida a linguagem WSat e construído um ambiente de execução para a mesma. Esta linguagem permite a escrita de testes, como os de aceitação e performance, com um alto nível de abstração e reuso, através da definição de tipos que representam componentes Web a serem testados.

Os componentes Web testados por programas WSat são páginas Web, formulários HTML, imagens, *links*, etc. Desta forma, WSat possui a capacidade de expressar explicitamente a estrutura da GUI dos sistemas testados. Por questões de espaço, apresentamos a seguir a linguagem WSat de forma sucinta, cujos detalhes podem ser vistos em artigos específicos sobre a mesma [2][3].

Podemos ver na Figura 1, a página inicial e a de resposta de um sistema de busca de documentos na Web. Para testar este sistema através de um programa WSat, a primeira tarefa a ser realizada é a descrição estrutural da sua GUI através da definição de tipos WSat, como visto a seguir.

```
static WebPage PagInicial {
    FormBusca formBusca;
    ...
}
```

O tipo `PagInicial` é definido para testar a página inicial do sistema de busca, verificando, entre outras coisas, a existência do formulário de busca representado pela propriedade `formBusca`. Este formulário deve satisfazer as características indicadas pelo tipo `FormBusca` visto a seguir.

Figura 1: Telas inicial e de resposta do sistema de busca.

```
WebForm FormBusca {
    name = "formBusca";
    method = "POST";
    EditBox {
        name = "palavras";
        value = "";
        ...
    } palavras;   ...
}
```

Propriedades de um tipo WSat definem as restrições sobre o conjunto de componentes Web denotado por este tipo. Na definição do tipo `FormBusca`, por exemplo, podemos ver o uso das propriedades `name` e `method`. Estas propriedades indicam respectivamente o nome e o método de submissão do formulário. Já a sua propriedade `palavras` representa o campos de texto do formulário utilizado para o preenchimento de palavras a serem procuradas pelo sistema.

Durante a definição de propriedades que representam parâmetros de formulários HTML, podemos fazer uso da cláusula `validate`. Esta cláusula permite indicar restrições sobre os valores que o parâmetro pode assumir. Esta informação é utilizada para geração de código de validação de parâmetros, como visto na Seção 5. Podemos ver a seguir, no trecho de código da definição da propriedade `palavras` do tipo `FormBusca`, o uso da cláusula `validate` indicando os valores válidos para as palavras chaves de busca.

```
EditBox {
    ...
    validate {
        type = String;
        optional = false;
        minLength = 3;
    }
} palavras;
```

Entre as restrições que podem ser utilizadas estão o tipo do parâmetro, seu tamanho mínimo e se ele é opcional. Outra informação utilizada para geração de código encontrada em programas WSat é a indicação de quais páginas não são geradas dinamicamente através de tecnologias como Servlets, CGI ou JSP. Para isto, utilizamos a palavra chave `static` na definição dos subtipos de `WebPage` que testam estas páginas estáticas, como no caso do tipo `PagInicial` mostrado anteriormente.

Por fim, definimos o tipo `PagResposta`, utilizado para testar a página de resposta do sistema:

```
WebPage PagResposta {
    title = "Resposta do Sistema de Busca";
}
```

Como podemos ver, o tipo `PagResposta` não faz uso de propriedades para verificar as informações da página Web geradas dinamicamente. O resultado da busca, que varia de acordo com os parâmetros da busca, é verificado durante descrição comportamental da GUI do sistema, realizada através da definição de casos de teste como o visto a seguir.

```
01: testCase buscar {
02:     String url = "http://www.sistemadebusca.com.br/";
03:     PagInicial pagInicial =
                            [PagInicial] getWebPage(url);
04:     FormBusca form = pagInicial.formBusca;
05:     form.palavras.value = "ufpe";
06:     PagResposta resp = [PagResposta] form.submit();
07:     WebLink link =
             resp.findWebLinkByURL("http://www.ufpe.br");
08: }
```

No caso de teste mostrado, podemos ver o uso do comando `getWebPage` na linha 3 para retornar um componente do tipo `WebPage` representando a página de URL indicada. Utilizamos o operador de transformação de tipo (`[PagInicial]`) para verificar se este componente satisfaz as características indicadas pelo tipo `PagInicial` e para transformá-lo em um componente do tipo `PagInicial`. Já as linhas 4 a 6 simulam digitação da palavra chave "`ufpe`" no formulário de busca, realiza a sua submissão através do serviço `submit` do formulário e transforma o tipo da página retornada de `WebPage` para `PagResposta`, verificando se as propriedades definidas pelo tipo são satisfeitas. Por fim, a linha 7 verifica se a página retornada possui em seu corpo o *link* "`http://www.ufpe.br`", indicando a correta resposta do sistema.

Como podemos notar, a utilização de tipos, propriedades e serviços definidos em WSat, assim como a separação entre a descrição estrutural e comportamental da GUI do sistema,

permitem um alto nível de abstração e legibilidade para o testador (programador de testes). A definição de tipos possibilita também um alto nível de reuso de código. O tipo `FormBusca`, por exemplo, pode ser utilizado na definição de quaisquer tipos WSat que testem páginas Web que possuam o formulário de busca. Já o código de casos de teste pode ser reusado através da definição de funções de teste.

As linguagens de teste atualmente disponíveis no mercado são baseadas geralmente em linguagens de programação, adicionando complexidade desnecessária, ou em XML, linguagem criada visando a definição de protocolos de comunicação entre programas. Embora de fácil aprendizado, o nível de abstração de programas escritos em XML é bastante limitado.

Visando um entendimento melhor de WSat, apresentamos a seguir a sua gramática resumida, mostrando algumas de suas principais construções.

```
WSatProgram = (TestComponentDec)+
TestComponentDec = WSatTypeDec | TestCaseDec | ...
WSatTypeDec = WebPageDec | WebFormDec | ...
TestCaseDec = "testCase" <IDENTIF> "{" Statement* "}"
Statement = AssertStat | WebServiceCall | ...
...
```

Como podemos observar, um programa WSat é formado por declarações de componentes de teste como tipos WSat e casos de teste. Os tipos WSat declarados podem ser subtipos de tipos como `WebPage` e `WebForm`. Casos de teste, por sua vez, são formados por comandos como os de atribuição e invocação de serviços Web.

Para a execução dos casos de teste, foi construiído um compilador de WSat para código Java [2][3]. De forma simplificada, o código Java gerado quando executado requisita páginas Web geradas pelo sistema em teste. Durante esta execução, é verificada a existência das propriedades definidas pelos tipos WSat nas páginas Web retornadas. Caso estas propriedades não sejam encontradas, a execução do caso de teste é interrompida indicando-se o motivo da falha na execução.

3 Processo de Teste

Para realizar testes como os de aceitação e de performance de forma eficaz e sem impactar a produtividade, definimos um processo de teste de sistemas Web e construímos ferramentas de apoio. Este processo de teste foi definido para ser encaixado no processo utilizado para o desenvolvimento de sistemas Web. Nele são definidos os papéis e atividades a serem realizadas pelos desenvolvedores. Também é definido o fluxo de execução das atividades, visto na Figura 2.

Algumas atividades são utilizadas como pontos de encaixe com o processo de desenvolvimento utilizado. Estas atividades encontram-se destacadas na Figura 2 através de linhas paralelas. Algumas atividades definidas por este processo de teste podem já existir no processo de desenvolvimento. Neste caso, as atividades podem ser unificadas.

Este processo de teste também possui pequenas atividades de implementação, devido a geração de código de sistema a partir dos programas de teste. Os detalhes sobre cada atividade definida neste processo são vistos a seguir.

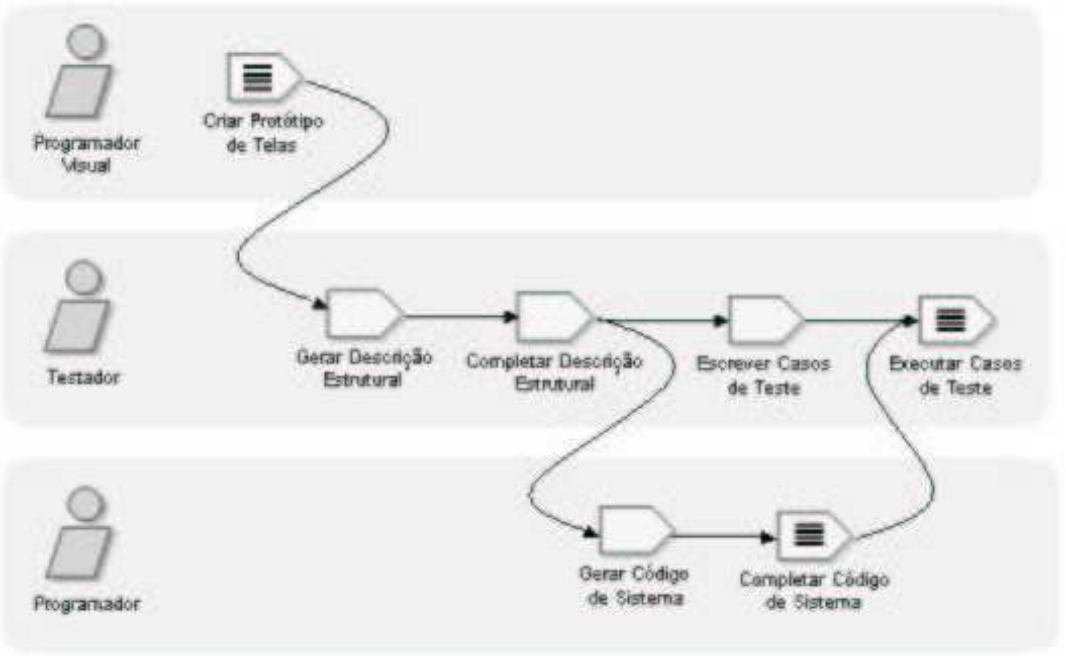

Figura 2: Fluxo de atividades do processo de teste.

3.1 Criar Protótipo de Telas

Esta atividade tem como objetivo criar um protótipo de telas HTML para o sistema Web a ser desenvolvido. Atividade de encaixe, ela pode ser conectada com atividades de elicitação ou validação de requisitos. O protótipo criado tem aqui o objetivo de ser utilizado como base para geração de código de teste.

Para a utilização eficaz do gerador de código de teste utilizado na atividade descrita a seguir, faz-se necessário que os protótipos de tela sejam desenvolvidos observando-se algumas considerações. Por exemplo, a nomenclatura utilizada para os componentes Web contidos nos arquivos HTML devem ser compatíveis com nomeclatura dos tipos WSat. Isto porque o gerador reutiliza a nomenclatura dos protótipos na geração de tipos e propriedades definidas em WSat.

3.2 Gerar Descrição Estrutural

O objetivo desta atividade é gerar automaticamente parte do programa de teste WSat, a partir do protótipo de telas construído para o sistema. Para isto, o testador utiliza o gerador de código de teste visto na Seção 4 para gerar o esqueleto dos tipos WSat a serem utilizados nos casos de teste do sistema. Os tipos WSat são gerados de acordo com os componentes Web encontrados no protótipo de telas.

Ao gerar de forma automática boa parte do código WSat que descreve a GUI do sistema, reduzimos o esforço de codificação dos testes. Com isto, reduzimos também o impacto na produtividade, a curto prazo, causado pelo tempo necessário para o desenvolvimento dos testes.

3.3 Completar Descrição Estrutural

Nessa atividade, o testador tem como objetivo tornar mais completa a descrição estrutural da GUI gerada automaticamente. Para isto, o testador adiciona códigos mais elaborados sobre a

estrutura da GUI que não puderam ser gerados de forma automática. Também são eliminadas redundâncias através do reuso do código utilizado na declaração de propriedades dos tipos WSat.

A nomenclatura gerada a partir dos protótipos de tela também pode ser melhorada, visando alcançar uma melhor legibilidade do código. O testador pode também remover trechos de código que foram gerados para testar componentes que não transmitem informações de negócio. Exemplos disto são as imagens utilizadas para auxiliar apenas a programação visual da página.

Visando a geração correta de código de sistema vista na próxima atividade, o testador também deve remover a palavra chave `static` da definição de subtipos de `WebPage` que testam páginas que, na verdade, serão geradas de forma dinâmcia pelo sistema em desenvolvimento. Além disto, ele deve adicionar cláusulas `validate` aos tipos que definem parâmetros de formulários HTML. Estas cláusulas indicam restrições sobre os possíveis valores a serem assumidos por estes parâmetros.

3.4 Gerar Código de Sistema

Esta atividade visa gerar trechos de código de sistema de forma automática. A partir da descrição da estrutura da GUI do sistema realizada pelos tipos WSat, trechos de código de sistema podem ser gerados pelo programador. Para isto, ele faz uso do gerador de código de sistema visto na Seção 5.

O código gerado serve de base de implementação do sistema, aumentado desta forma a produtividade a curto prazo. O gerador de código de sistema deve ser especializado para gerar tipos de códigos compatíveis com o ambiente de desenvolvimento utilizado.

3.5 Completar Código de Sistema

Visando obter a implementação de uma versão inicial do sistema com o comportamento definido pelo protótipo de telas, esta atividade realiza ajustes no código de sistema gerado automaticamente para que os mesmos possam ser executados.

Para isto, o programador realiza pequenos ajustes em arquivos ou *templates* HTML, corrigindo *links* que possam por ventura ter sido quebrados durante a geração de código. *Handlers* também podem ser neste momento eliminados devido ao padrão de projeto *Web Handlers* permitir facilmente o reuso de *handlers* durante a composição dos serviços Web a serem fornecidos.

Atividade de encaixe, ela deve ser conectada à atividades do processo de desenvolvimento que realizam a implementação das funcionalidades do sistema. Estas atividades de implementação devem tomar como base o código de sistema gerado.

3.6 Escrever Casos de Teste

Após a definição dos tipos WSat a serem utilizados nos testes, o testador está apto a escrever em programas WSat os casos de teste para cada funcionalidade do sistema. O desenvolvimento destes casos de teste é baseado nos requisitos do sistema, definidos em atividades do processo de desenvolvimento. Estes casos de teste devem testar o código escrito pelo programador durante as atividades de implementação do sistema.

O processo de teste mostrado neste trabalho não leva em consideração outros tipos de testes que não os suportados pela linguagem WSat. Um exemplo disto são os testes de unidade, que devem ser realizados em atividades definidas pelo processo de desenvolvimento utilizado.

3.7 Executar Casos de Teste

Esta atividade tem como objetivo executar os casos de teste do sistema Web, identificando possíveis defeitos inseridos no sistema durante a programação do mesmo. Para a execução destes testes, fazemos uso do ambiente de execução implementado para WSat.

Não foram realizados neste trabalho, estudos sobre os momentos mais adequados para a programaçõ dos casos de teste. Entretanto, as principais metodologias existentes na engenharia de software já definem as fases para a realização dos testes.

4 Gerador de Código de Teste

Durante a análise de requisitos de sistemas Web, é comum a utilização de protótipos de telas HTML construídos por programadores visuais (*Web Designers*). Estes protótipos possuem um baixo custo de desenvolvimento e permitem um melhor entendimento e validação dos requisitos através de simulações de uso do sistema próximas do seu funcionamento real. Ao se comparar estes protótipos e programas WSat escritos para um mesmo sistema, é possível encontrar no código fonte dos protótipos parte das informações escritas nos programas WSat. Por exemplo, informações sobre formulários HTML, imagens e links vistas nos tipos WSat podem ser encontradas nos arquivos HTML dos protótipos de telas, mesmo que com nível de abstração reduzido.

Visando reduzir o esforço necessário para programação dos testes, foi construído um gerador de código de teste capaz de gerar código WSat a partir de protótipos de telas HTML. Durante esta seção, discutimos os tipos de código WSat que são gerados automaticamente.

Inicialmente, para cada arquivo HTML do protótipo de telas é gerado um novo tipo WSat, mais precisamente um subtipo de WebPage. O arquivo `paginaInicial.html` que representa a página inicial do sistema de busca mostrada na Figura 1, por exemplo, resulta na geração do esqueleto do tipo `PagInicial` mostrado na Seção 2. Como podemos perceber, os tipos gerados para cada arquivo HTML possuem nome igual ao arquivo, sem sua extensão, e com a primeira letra maiúscula.

A partir do conteúdo dos arquivos HTML do protótipo de telas, geramos propriedades nos tipos WSat que representam a existência de subcomponentes. A partir do código HTML que declara o formulário de busca no protótipo do sistema, por exemplo, geramos a propriedade `formBusca` vista no tipo `PaginaInicial`.

As informações utilizadas na geração de código são extraídas dos atributos de marcadores HTML. No caso dos formulários HTML, são analisados atributos como `name` e `method` do marcador HTML <`form`>. Para identificar os parâmetros dos formulários, são analisados também os atributos de outros marcadores, como o <`input`> e o <`select`>. A geração de propriedades para outros componentes Web, como por exemplo *links* e imagens, é realizada da mesma forma.

Existem informações que podem ser adicionadas a programas WSat para serem utilizadas pelo gerador de código de sistema visto na Seção 5. Parte destas informações são geradas automaticamente pelo gerador de código de teste. Por exemplo, geramos a propriedade `template` como visto a seguir para indicar o caminho do arquivo HTML que originou a definição deste tipo.

```
static WebPage PaginaInicial {
    template = "c:\wsat\prototipo\paginaInicial.html";  ...
}
```

O gerador de código de teste utiliza a palavra chave `static` em todos os subtipos de `WebPage` gerados, devendo o testador remover esta palavra chave dos tipos cujas páginas testadas são geradas dinamicamente.

5 Gerador de Código de Sistema Web

O gerador de código de teste mostrado anteriormente permite uma redução no tempo para definição dos tipos WSat utilizados nos casos de teste. Entretanto, o código destes casos de testes ainda é escrito de forma totalmente manual.

Visando compensar parte do tempo gasto com a criação dos casos de teste e aumentar a produtividade, foi construído um gerador de código de sistema. Só foi possível implementar este gerador devido às características de WSat de expressar explicitamente a estrutura da GUI do sistema e de possuir informações auxiliares utilizadas para a geração de código. Nesta seção, mostramos tipos de código de sistemas Web que podem ser gerados automaticamente a partir de seus programas de teste escritos em WSat.

A geração de código de sistema é, em parte, dependente do ambiente utilizado para o desenvolvimento destes sistemas. O código deve ser gerado, por exemplo, de acordo com a linguagem de programação utilizada. Desta forma, para criar um gerador de código de sistema Web que seus usuário possam tirar o máximo proveito, faz-se necessário especializá-lo para gerar código de acordo com o ambiente de desenvolvimento utilizado.

A especialização do gerador de código de sistema é realizada através da implementação de *visitors* [8] que percorrem árvores sintáticas dos programas WSat. Cada um destes *visitors* possui uma funcionalidade específica, como por exemplo a de gerar o esqueleto de classes que atendem às requisições Web. A seguir, mostramos os principais tipos de código de sistema gerados automaticamente por este gerador.

5.1 *Web Handlers*

Páginas dinâmicas de sistemas Web podem ser implementadas de diversas formas, como por exemplo através de Servlets, CGI, JSP e *Web Handlers* [1]. Para implementação deste gerador, consideramos o uso do padrão de projeto *Web Handlers*. De forma geral, com a utilização deste padrão, cada página dinâmica do sistema é associada a um *handlers* de processamento e a um outro de apresentação.

Para cada um dos subtipos de WebPage que testam páginas dinâmicas, ou seja, subtipos declarados sem a cláusula WSat `static`, um par de *handlers* é gerado para implementar a página Web do sistema a ser testada por este tipo. O primeiro destes *handlers* é o de processamento, responsável pela execução da funcionalidade requerida a uma dada página do sistema. O trecho de código do *handler* de processamento do sistema de busca gerado para o tipo `PagResposta` é visto a seguir:

```
public class PagRespostaHandlerProcessamento
                            extends HandlerProcessamento {
    public void processar(HttpServletRequest request,
                          HttpServletResponse response)
                       throws ProcessamentoException {
    }
}
```

Este *handler* herda da classe `HandlerProcessamento` definida pelo padrão *Web Handlers* e possui o método abstrato `processar`, responsável por executar a funcionalidade do sistema requerida.

O segundo *handler* gerado é o *handler* de apresentação, responsável por montar a apresentação do resultado obtido pela execução do *handler* de apresentação. Para o tipo WSat `PagResposta`, o seguinte *handler* de apresentação é gerado:

```
public class PagRespostaHandlerApresentacao
                            extends HandlerApresentacao {
    public void apresentar(HttpServletRequest request,
                        HttpServletResponse response)
                              throws ApresentacaoException {
        ...
    }
}
```

Como podemos ver, o *handler* gerado herda da classe `HandlerApresentacao` definida no padrão *Web Handlers* e possui o método abstrato `apresentar`, utilizado para montar a página de resposta para as requisições de serviços Web. Podemos notar que o código gerado para os *handlers* não é substancial. Entretanto, como o número de *handlers* necessários para implementar o sistema pode ser grande, esta geração de código se torna bastante interessante.

5.2 Código de Validação de Parâmetros

É comum que requisições de sistemas Web necessitem de parâmetros para a sua execução. O sistema de busca mostrado necessita, por exemplo, de palavras chave para realizar a busca. Visando obter sistemas Web mais robustos, são escritos códigos para validar os valores destes parâmetros antes de ser utilizá-los.

Para validar os parâmetros recebidos, podemos utilizar código JavaScript e esquemas XML [13]. JavaScript é utilizado para validar os dados no próprio navegador Web do cliente, melhorando assim a performance do sistema. Esquemas XML podem descrever regras de validação para os valores destes parâmetros e são utilizados para validar os dados enviados no servidor Web.

A seguir podemos ver parte do esquema XML gerado para validar os parâmetros do formulário HTML do sistema de busca.

```
<?xml version="1.0"?>
<schema>
    <attribute name="palavras">
        <simpleType base="string" optional="false"
                    minLength=3 />
    </attribute>   ...
</schema>
```

Como podemos notar, este arquivo contém informações, sobre o tipo do parâmetro, seu tamanha mínimo, etc. Este tipo de informação é extraída da cláusula `validate` encontrada na definição de tipos WSat que representam parâmetros de formulários HTML.

Para cada formulário HTML do sistema é gerado um arquivo de esquema XML correspondente, com validações de acordo com as informações encontradas nas cláusulas `validate` de

seus parâmetros. Parâmetros que não utilizam esta cláusula são representados nos arquivos XML como parâmetros sem nenhuma restrição de validação.

A validação dos dados pelo servidor é essencial, já que o código de validação em JavaScript enviado ao cliente pode ser facilmente desativado. Além disto, como a geração dos arquivos JavaScript e XML é feita de forma similar, apenas a geração dos arquivos XML foi abordada na implementação deste gerador.

5.3 Testes Unitários de *Handlers*

A realização de testes unitários é indicada para todas as classes do sistema, inclusive para as que implementam *handlers* de requisições de serviços Web. Para cada *handler* gerado para o sistema, geramos também o esqueleto de uma classe de teste de acordo com o *framework* JUnit [7]. A seguir mostramos da classe gerada para realizar testes unitários na classe `PagRespostaHandlerProcessamento` mostrada anteriormente:

```
public class TestPagRespostaHandlerProcessamento
                                        extends TestCase {
    private PagRespostaHandlerProcessamento handler;
    private HttpServletRequest  req;
    private HttpServletResponse resp;   ...
    public setUp() {
        handler = new PagRespostaHandlerProcessamento();
        req  = new HttpServletRequestImpl();
        resp = new HttpServletResponseImpl();
    }
    public void testProcessar() {
        req.put("paramExemplo1", "valorExemplo1");
        handler.processar(req, resp);
    } ...
}
```

Como podemos notar, o código para inicialização do ambiente de teste é parcialmente gerado no método `setUp`, com a instanciação do *handler* e de seus parâmetros `req` e `resp`. O código de teste a ser executado para o *handler* deve ser escrito pelo testador dentro do método `testProcessar` da classe de teste gerada. Para simular uma requisição Web, por exemplo, podemos definir os seus parâmetros utilizando o método `put`, como mostrado no exemplo. O resultado do processamento pode ser verificado através das variáveis `req` e `resp`.

5.4 Outros Tipos de Códigos Gerados

A API FreeMarker [9] é utilizada para construção de páginas Web dinâmicas, permitindo a separação entre código Java e código HTML. Para a sua utilização com os *handlers* de apresentação do sistema, associamos a cada um deles um *template* HTML. *Templates* HTML são arquivos com variáveis que serão substituídas por valores gerados pelo sistema em tempo de execução.

A partir da propriedade `template` de subtipos de `WebPage`, podemos copiar os arquivos HTML do protótipo de tela que os geraram para utilizá-los como base para o desenvolvimento das páginas estáticas (arquivos HTML fixos) ou dinâmicas (*templates* dos *handlers*) do sistema. O código gerado para cada *handler* de apresentação possui uma implementação básica

do método `apresentar` que gera uma página de resposta de acordo com seu *template* associado durante a geração de código.

Para o funcionamento correto do sistema, é necessário também indicar para cada página dinâmica (serviço Web) do sistema o par de *handlers* que o implementa. Isto é feito através de um arquivo XML de configuração de serviços. Para cada tipo WSat que testa páginas Web dinâmicas, geramos neste arquivo de configuração um serviço com o nome do tipo WSat e associado ao par de *handlers* gerado para implementá-lo. Os *handlers* também necessitam ser configurados adequadamente através de parâmetros contidos em um arquivos de configuração de *handlers*. Os parâmetros básicos para o funcionamento de cada um dos *handlers* também são gerados automaticamente.

Através da geração de código de sistema discutida, nós possibilitamos aos desenvolvedores a compilação e execução preliminar do sistema Web sem necessidade de muita programação. A partir de pequenos ajustes, o esqueleto gerado da GUI do sistema se comporta da mesma maneira que o protótipo de telas, restando aos desenvolvedores implementar essencialmente camadas inferiores a da GUI do sistema.

6 O Experimento Realizado

Para avaliar os resultados obtidos com a definição do processo de teste e dos geradores de código desenvolvidos, realizamos um experimento através do desenvolvimento de um sistema Web de pequeno porte de uma empresa real, cujos nomes não podem ser divulgados aqui por motivos estratégicos. Um dos motivos da escolha deste sistema foi o uso de tecnologias diretamente relacionadas aos geradores de código construídos neste trabalho, como por exemplo as linguagens Java e HTML. A equipe de desenvolvimento deste projeto foi formada pelo primeiro autor e por mais dois programadores Java experientes. Para realizar o experimento, decidimos implementar o sistema Web escolhido de duas formas:

- Proj. A: Utilizando a metodologia convencional da empresa, sem a utilização do processo e ferramentas de teste aqui desenvolvidas.
- Proj. B: Utilizando a metodologia convencional da empresa, mas adicionando-se a utilização do processo e ferramentas definidos por este trabalho.

Devido às limitações de recursos e de tempo, a mesma equipe de desenvolvimento implementou os dois projetos. Códigos de camadas inferiores a da GUI do sistema foram reaproveitados. Os resultados finais obtidos com este reuso do código não são impactados, já que o desenvolvimento dos códigos reutilizados não são influenciados pela utilização de WSat e das ferramentas desenvolvidas neste trabalho. As tarefas executadas foram divididas de forma a evitar que programadores escrevessem o mesmo código nos dois projetos.

Durante o desenvolvimento dos Projetos A e B foram coletados dados para serem analisados posteriormente. As principais informações registradas durante as atividades foram o nome da atividade, sua data de realização e tempo gasto, assim como o número de linhas de código escritas e geradas automaticamente.

Todos estes dados foram então coletados e armazenados em planilhas. Com o reuso de código do projeto A no projeto B, algumas das planilhas do projeto A foram reutilizadas no projeto B. Este fato não é um problema para o que queremos analisar, já que estes dados não sofrem influência do uso de WSat e das ferramentas aqui desenvolvidas.

Para guiar uma análise comparativa dos resultados obtidos, foram selecionadas métricas de qualidade e produtividade. As métricas para medir fatores de qualidade de software indicam os benefícios e os problemas obtidos nos sistemas Web desenvolvidos com a utilização de WSat e das ferramentas aqui construídas. Exemplos deste tipo de métrica são a corretude do sistema e a facilidade de leitura do código gerado. Já as métricas utilizadas para medir a produtividade indicam o impacto na produtividade obtido no desenvolvimento de tais sistemas, como por exemplo o esforço de codificação.

Com relação aos fatores de qualidade do sistema desenvolvido, obtivemos resultados interessantes como a redução do número de erros. O número de erros encontrados no Projeto B foi 11,53% menor que o número de erros no Projeto A. Alguns destes erros foram removidos simplesmente por parte do código ter sido gerado automaticamente. Já a impressão dos programadores com relação à geração de código também foi boa. O código Java gerado, por exemplo, mostrou-se semelhante ao escrito no Projeto A. Com relação ao código WSat gerado, foram encontradas dificuldades para o seu entendimento e correção devido à pobre nomenclatura que foi utilizada no protótipo de telas HTML.

Analisando o esforço de codificação dos testes, foi verificado que o número de linhas da descrição comportamental do caso de teste ficou entre 15% e 30% do número de linhas utilizadas para a descrição estrutural da GUI do sistema. Isto representa algo entre 13% e 24% do número de linhas total de código de teste. Como parte da descrição estrutural da GUI do sistema é parcialmente gerada a partir de protótipos de telas, uma boa parte do código WSat é automaticamente escrito, diminuindo consideravelmente o esforço para codificação dos testes.

Entretanto, não foi encontrado nenhum padrão com relação ao tempo de criação destes testes. Isto porque o tempo de desenvolvimento mostrou-se depender dos tipos e não do número de componentes Web envolvidos. Componentes Web do tipo imagens e *links* são automaticamente testados pela descrição estrutural da GUI, diferentemente de tabelas que em geral não são representadas na descrição estrutural por poder ter seu tamanho e conteúdo diferentes em cada resposta.

Com relação ao tempo necessário para a implementação de um primeiro protótipo usável, a utilização dos geradores de código resultou em uma redução de 3% no tempo gasto pelo Projeto A. Este ganho de tempo é pequeno, tornando-se mais significativo quando consideramos que a maior parte do trabalho com a validação dos dados recebidos pelos *handlers* já foi realizado neste ponto no Projeto B, pois o mesmo é feito durante a definição dos tipos em WSat.

Com relação ao esforço total de desenvolvimento, ou seja, dos testes e do sistema, houve uma redução de 5,45% em relação ao tempo de desenvolvimento e um aumento de 1,73% do número de linhas de código escritas. Também obtivemos a geração de 4,52% do total das linhas de código. Este valor se torna bastante significativo quando se observa o tipo dos códigos que foram gerados. Mesmo que sem grandes complexidades, estes códigos fornecem em geral um trabalho repetitivo e entediante.

7 Conclusões

A principal contribuição deste trabalho é a definição de um processo para realização eficaz de testes de sistemas Web. O processo definido melhora a produtividade com a realização dos testes em sistemas web através do uso de geradores de código. Este processo de teste foi definido para ser encaixado em processos maiores que abrangem todas as etapas a serem cumpridas durante o desenvolvimento de sistemas Web.

Para suportar a programação de testes de sistemas Web como os de aceitação e de per-

formance, definimos a linguagem WSat e um ambiente de execução para a mesma. Através de programas escritos em WSat, podemos verficar o conteúdo das páginas HTML de resposta retornadas pelos sistemas testados. Componentes Web mais elaborados como Applets, Flash, JavaScript e XML ainda não são suportados pela linguagem.

Contrastando com outras abordagens, esta linguagem permite a escrita de programas com um alto nível de abstração através da definição de tipos que representam componentes Web. Em programas WSat, podemos também expressar informações que permitem explorar melhor o desenvolvimento de geradores de código de sistema.

Devido às características especiais de WSat, foi possível implementar dois geradores de código para dar suporte ao processo de teste definido e para diminuir o impacto causado na produtividade, a curto prazo, pela programação dos testes. O primeiro gerador é utilizado para, a partir de protótipos de telas HTML, gerar parte do código WSat que descreve a estrutura estática da GUI do sistema. Com isto, reduzimos o esforço da programação dos testes, podendo o programador de teste se concentrar na parte dinâmica do sistema. O segundo gerador implementado é utilizado para gerar, entre outras coisas, trechos de código Java do sistema testado a partir de programas WSat. Desta forma, conseguirmos melhorar também a produtividade do desenvolvimento dos sistemas.

Além de aumentar a produtividade, os geradores de código desenvolvidos auxiliam a utilização da prática *Test First Design* [6], indicada pela metodologia *Extreme Programming* [4], onde os testes são criados antes da implementação do código testado.

Para avaliar o trabalho desenvolvido, realizamos um experimento com o processo de teste, a linguagem WSat e as ferramentas construídas. Neste experimento, verificamos o aumento da corretude do sistema com a utilização dos testes escritos em WSat. Também foi possível verificar o aumento de produtividade. De fato, o ganho com a geração de trecho de códigos de sistema mostrou ter sido suficiente para a programação dos programas WSat. Entretanto, se a cobertura do sistema pelos testes fosse ampliada com um número grande de casos de teste, o ganho de tempo com a geração de código não seria suficiente para compensar totalmente o tempo gasto com a definição dos casos de teste.

Para provar os benefícios gerados pela utilização do processo de teste e ferramentas de apoio definidas, faz-se necessária a realização de um número maior de experimentos. Entretanto, através do experimento realizado pudemos verificar a ordem de magnitude da perda de produtividade, a curto prazo, obtida com a utilização do processo de teste definido. O fato de obtermos uma redução no tempo de desenvolvimento ao invés do acréscimo que era esperado sugere que, mesmo obtendo resultados inferiores com a realização de outros experimentos, não devemos verificar impactos negativos significantes na produtividade com a realização dos testes.

Como trabalhos futuros, podemos explorar as associações entre páginas Web representadas por como *links* e ações de formulários HTML. Estudando mais profundamente estas associações, podemos verificar se as mesmas podem permitir a geração de outros tipos de código que não os já implementados. Um exemplo disto é a geração de trechos de casos de teste.

Apesar de não ser uma atividade de pesquisa, para tornar viável a utilização das ferramentas desenvolvidas faz-se necessário o desenvolvimento de interfaces mais amigáveis. Além disto, podemos otimizar e remover limitações das ferramentas.

Como trabalhos relacionados, temos a linguagem TestTalk [11] que visa a portabilidade dos testes, independente de linguagem e ambiente de desenvolvimento utilizados. Mesmo permitindo a definição de termos de domínio, seu nível de abstração é inferior ao obtido com utilização de tipos WSat. Linguagens de teste disponíveis no mercado, como a Canoo Web Test [5], são baseadas em XML ou em linguagens de programação, adicionando assim complexidade

desnecessária e prejudicando o seu nível de abstração. Uma comparação mais detalhada entre WSat e outras linguagens existentes pode ser encontrada em artigos que tratam especificamente da linguagem [2][3].

Outro trabalho relacionado é a geração de testes funcionais [12] através de modelagem computacional com *X-machines*, onde são identificados os eventos e entradas que produzem saídas observáveis no sistema. Estas informações são utilizadas para guiar a geração de casos de teste.

Agradecimentos

Os autores agradecem os comentários recebidos dos julgadores deste artigo, o que contribuiu para melhorar a qualidade deste trabalho.

Referências

[1] Gibeon Aquino and Paulo Borba. Web Handlers. In *First Latin American Conference on Pattern Languages of Programming, Sugarloaf PLoP*, Rio de Janeiro, Brazil, 3th–5th October 2001.

[2] Eduardo Aranha and Paulo Borba. Uma Linguagem para Testes de Aceitação de Sistemas Web. In *VI Simpósio Brasileiro de Linguagens de Programação*, Rio de Janeiro, Brasil, 5 a 7 de Junho 2002. Aceito para publicação.

[3] Eduardo Aranha and Paulo Borba. Web Systems Acceptance Tests and Code Generation. In *Workshop on Testing in XP. Third International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Alghero, Sardinia, Italy, May 27 2002. Accepted for publication.

[4] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[5] Canoo Engineering. Canoo WebTest, 2002. http://webtest.canoo.com.

[6] Michael Feathers. Test First Design, 2000. http://www.xprogramming.com/xpmag/test_first_intro.htm.

[7] Erich Gamma and Kent Beck. JUnit. http://www.junit.org.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, October 1994.

[9] Benjamin Geer and Mike Bayer. FreeMarker. http://freemarker.sourceforge.net.

[10] Ivar Jacobson. *The Unified Software Development Process*. Addison–Wesley, first edition, 1999.

[11] Chang Liu and Debra J. Richardson. TestTalk: A Comprehensive Testing Language. In *the 14th IEEE International Conference on Automated Software Engineering, Doctoral Symposium*, Cocoa Beach, Florida, USA, October 1999.

[12] M. Gheorghe M. Holcombe, K. Bogdanov. Functional Test Generation for Extreme Programming. In *2nd International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Alghero, Sardinia, Italy, 20-23 May 2001.

[13] Brett McLaughlin. Validation with Java and XML Schema. http://www.javaworld.com/javaworld/jw-09-2000/jw-0908-validation.html.