Universidade Federal de Santa Maria

Centro de Tecnologia

Curso de Ciência da Computação

Busca de Oportunidades de Refatoração em Aplicações Web

Trabalho de Graduação

Santa Maria, RS, Brasil

2011

Busca de Oportunidades de Refatoração em Aplicações Web

por

Pablo Lima Flores

Trabalho de Graduação apresentado ao curso de Ciência da Computação da Universidade Federal (UFSM, RS), como requisito parcial para obtenção do grau de

Bacharel em Ciência da Computação

Orientador: Prof Dr Eduardo Kessler Piveta

Trabalho de Graduação, N^{o} 337

Santa Maria, RS, Brasil

2011

Universidade Federal de Santa Maria

Centro de Tecnologia

Curso de Ciência da Computação

A comissão Examinadora, abaixo assinada,

aprova o Trabalho de Graduação

Busca de Oportunidades de Refatoração em Aplicações Web

Elaborado por

Pablo Lima Flores

como requisito parcial para obtenção do grau de

Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

_____

Prof Dr Eduardo Kessler Piveta
(Presidente/Orientador)

_____

Prof Dr Marcia Pasin

_____

Prof Dr Giovani Rubert Librelotto

Santa Maria, 16 de dezembro de 2011

AGRADECIMENTOS

Primeiramente agradeço a Deus pela força, paciência e resignação que me fizeram chegar a este momento crucial de minha vida.

Agradeço aos professores, estagiários, funcionários e colegas com os quais tive a oportunidade de conviver nesses últimos anos.

Sou grato também aos meus parentes, tios, primos e sobrinhos pelo voto de confiança, em especial minha mãe Maria de Fátima, que me ama se orgulha de mim, minha irmã Marilu que torce pelo meu sucesso e minha irmã Michelle que está me auxiliando, dando-me suporte que, sem este, dificilmente eu teria condições de seguir o este curso e chegar até aqui.

Meus amigos, os verdadeiros, que me ajudaram, me apoiando, aconselhando, dividindo comigo os momentos felizes ou me ajudando a suportar meus momentos mais difíceis.

E por fim, direciono este agradecimento a duas pessoas que infelizmente não estão mais entre nós, meu irmão Diego (1982-2007) e o meu pai Aluízio (1955-2008). Entristece-me muito o fato de que não posso compartilhar com vocês fisicamente este momento, mas esta conquista, com certeza, é para vocês.

Obrigado a vocês, por tudo.

“Que os vossos esforços desafiem as impossibilidades, lembrai-vos de que as grandes coisas do homem foram conquistadas do que parecia impossível.”

- Charles Chaplin

RESUMO

Trabalho de Graduação

Curso de Ciência da Computação

Universidade Federal de Santa Maria

Busca de Oportunidades de Refatoração em Aplicações Web

Elaborado por: Pablo Lima Flores

Orientador: Prof Dr Eduardo Kessler Piveta

Local e data da defesa: Santa Maria, 16/12/2011

Este trabalho objetiva a criação de buscas XQuery em páginas codificadas em XHTML para a detecção de possíveis oportunidades de refatoração, visando melhorar a estrutura do código fonte, bem como o desempenho de aplicações Web.

Refatoração é o processo de alterar um programa ou sistema de software para aprimorar a estrutura interna do código sem alterar seu comportamento externo observável. A refatoração de código em aplicações Web está cada vez mais inserida no cotidiano dos programadores, empresas de desenvolvimento de software e quaisquer empresas que possuem algum tipo de sistema o qual se deseja aumentar o número de funcionalidades ou melhorar o desempenho do mesmo. Uma das áreas de refatoração mais importantes de se tratar nos sistemas Web é a refatoração do código HTML, pois é a linguagem de marcação mais utilizada para produção de páginas Web.

Palavras-chave: Refatoração, HTML, Web

ABSTRACT

Trabalho de Graduação

Graduate Program in Computer Science

Universidade Federal de Santa Maria

Search for Refactoring Opportunities in Web Applications

Author: Pablo Lima Flores

Advisor: Prof Dr Eduardo Kessler Piveta

This work aims at creating XQuery queries encoded in XHTML pages for the detection of possible opportunities for refactoring, to improve the structure of the source code as well as the performance of Web applications.

Refactoring is the process of changing a program or software system to improve the internal structure of code without changing its observable external behavior. Refactoring code in Web applications is increasingly embedded in everyday programmers, software development companies and any companies that have some kind of system which you want to increase the number of features or improve its performance. One of the most important areas of refactoring of Web systems is dealt with in the HTML code refactoring, it is the markup language most often used to produce Web pages.

Key-words: Refactoring, HTML, Web.

Lista de figuras

Lista de Abreviaturas

CPU - Central Processing Unit

CSS – Cascading Style Sheets

HD - Hard Disk

HTML – Hypertext Markup Language

PC - Personal Computer

RAM - Random Access Memory

SQL – Structure Query Language

URL – Uniform Resource Locator

W3C – World Wide Web Consortium

WHATWG – Web Hypertext Application Technology Group

XHTML – Extensible Hypertext Markup Language

XML – Extensible Markup Language

Sumário

1. INTRODUÇÃO

Com a evolução e a massificação da internet em todos os setores da sociedade, a importância das aplicações na Web aumentou nos últimos anos. E com esse acréscimo, e com o melhoramento e a padronização das normas que regem a W3C, os sistemas passaram a ser desenvolvidos de forma que haja uma facilidade de entendimento para os desenvolvedores e de relativamente fácil inclusão de novas funcionalidades.

Tratando-se de aplicações Web, uma das áreas abordadas é à camada de apresentação, normalmente no formato de páginas Web. Nestas, a boa formatação do código fonte é um importante fator que pode auxiliar a melhorar fatores de qualidade como acessibilidade, legibilidade e no desempenho global do sistema.

Sabendo-se que os sistemas mais antigos ou com menor preocupação em modificações futuras e melhora no desempenho podem apresentar problemas estruturais na criação dos códigos-fonte de suas páginas. Para isto, é interessante que existam programas para identificar estes problemas e se necessário, que o código seja reestruturado.

1.1 OBJETIVO

Este trabalho tem como objetivo estudar e desenvolver técnicas para a busca e detecção de oportunidades de refatoração em códigos XHTML de sistemas Web, sugerindo possíveis soluções de refatoração nesses casos.

Alguns exemplos de refatorações para as quais serão buscadas oportunidades de sua aplicação neste trabalho são:

- Substituir *tags* como *i*, *b* para CSS;
- Substituir layouts baseados em tabelas;
- Substituir GET para POST ou vice-versa;
- Adicionar tipos de formulários WEB 2.0;
- Substituir flash por HTML;
- Substituir formulários de contato por MAILTO;
- Etc.

1.2 JUSTIFICATIVA

Os sistemas Web mais antigos podem possuir limitações (chamadas de “bad smells”) que são algumas características que podem ser melhoradas num sistema através de refatoração de código.

Este trabalho terá como enfoque principal buscar oportunidades de refatoração em código XHTML de sistemas Web, tendo como base um catálogo de refatorações para sites Web.

1.3 ESTRUTURA DA MONOGRAFIA

No capítulo 2 é apresentada uma revisão bibliográfica, mostrando algumas definições de termos, conceito de refatoração e suas aplicações, e linguagens de marcação e consulta que serão abordadas neste trabalho, como por exemplo, HTML, XML, XHTML, XPath e XQuery. Já no capítulo 3 é abordado o tema principal do trabalho que é Busca de Oportunidades de Refatoração. Primeiramente será apresentada uma introdução, havendo a contextualização da refatoração através do conceito de papéis, artefatos e atividades. Em seguida, será mostrada uma lista de refatorações que serão trabalhadas com os seus respectivos artefatos papéis, artefatos e atividades. No capítulo 4 é realizado um estudo de caso com um site desenvolvido em XHTML. Neste será realizados todas as buscas estudadas anteriormente e apresentado os resultados. No capítulo 5 são apresentados alguns trabalhos relacionados à refatoração como processo geral. No capítulo 6 é realizada a conclusão deste trabalho.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta a revisão bibliográfica deste trabalho. Primeiramente serão descritas as principais linguagens de marcação de hipertexto que são utilizadas atualmente na criação das páginas Web, o HTML e o XHTML, com sua definição, um breve histórico, e suas principais características. Após será detalhado o que é e qual a função da refatoração dos sistemas de software em geral e também mais especificamente, refatoração em código HTML e XHTML em sistemas Web, que é o foco deste trabalho.

Seguindo o capítulo, serão apresentadas algumas linguagens de consulta em documentos XML ou XHTML. Primeiro será abordado sobre XPath com definições e exemplo e depois sobre XQuery, que será a linguagem de consulta adotada para a criação dos aplicativos de busca de oportunidades de refatoração em documentos XHTML.

2.1 HTML

HTML (*HyperText Markup Language*) é uma linguagem de marcação padrão da W3C voltada para criação de páginas Web, fornecendo mecanismos para a estruturação e apresentação visual de documentos a serem interpretados pelos navegadores Web [W3C HTML, 2011].

Segundo [W3C HTML, 2011], a tecnologia HTML é fruto da união de dois padrões mais antigos, o HyTime (Hypermedia/Time-based Document Structuring Language) e o SGML (*Standart Generalized Markup Language)*.

O HyTime, segundo [NEWCOMB, 1991] é um padrão no qual um documento é visto como um conjunto de eventos concorrentes e dependentes do tempo e estes conectados por hiper-ligações. Para [HERWINJNEN, 1994], o SGML é, basicamente, um padrão relacionado puramente à formatação de textos, o qual se tornou interessante para transformar documentos em hiper-objetos e também para descrever suas ligações. De acordo com Javier [JAVIER, 2008], o primeiro documento formal com a descrição de HTML foi publicado em 1991 sob o nome *"Tags HTML"*

Os documentos HTML são construídos através do conceito de *tags* ou etiquetas As *tags* apresentam a seguinte estrutura:

<tag>*Conteúdo da tag... </**tag**>***

Como regra geral, as *tags* formam blocos que iniciam com uma instrução e terminam com a mesma instrução, com o acréscimo da barra no início "/". Desta forma as *tags* têm a função de definir a formatação de parte do documento, assim é feita a marcação da onde se inicia e onde se termina o texto com a formatação especificada por ela. Também há o conceito de *tag* vazia, que tem como função apenas inserir algum elemento ao documento:

<tag></tag> **ou** ***<tag/>***

A linguagem teve uma grande evolução desde a sua versão inicial, o HTML 1.0, de 1991 até a versão atual, o HTML 5 sendo desenvolvido atualmente. Nas primeiras versões eram definidas regras sintáticas com alto nível de flexibilidade, o que facilitava os que tinham maior dificuldade com a publicação Web na época. Atualmente esta flexibilidade praticamente não existe, o que torna as regras de criação de documentos HTML mais rígidas, mas com documentos mais bem formatados.

Atualmente a W3C juntamente com a WHATWG (Web Hypertext Application Technology Working Group) estão trabalhando no HTML5, que segundo [W3C HTML5, 2011] a primeira especificação foi anunciada em 2008.

O HTML5 que tem como um dos principais objetivos facilitar a manipulação dos seus elementos, possibilitando ao desenvolvedor modificar as características dos objetos de forma não intrusiva de maneira eu fique transparente ao usuário final.

A linguagem oferece importantes mudanças em relação às versões anteriores, por exemplo, oferecendo ferramentas para CSS e Javascript, criando novas *tags* e alterando a função de outras, Outra promessa é a provável falta de necessidade de se instalar plug-ins específicos para executar diversos tipos de padrão de vídeo, por exemplo.

2.2 XHTML

A linguagem XHTML (*eXtensible HyperText Markup Language*) é também uma linguagem de marcação padrão da W3C, cujo o formato é a combinação das linguagens HTML e XML (*eXtensible Markup Language*).

A linguagem XML [W3C XML, 2011] foi desenvolvida no ano de 1996 pela W3C e é uma linguagem, assim como o HTML, derivada do SGML, e tem como principal característica o formato de texto simples e flexível. Ela foi desenvolvida com o intuito de ser uma metalinguagem (linguagem que serve para descrever outras linguagens) flexível, mas formal permitindo a troca de dados entre instituições através da internet. O XML, diferente do HTML não possui nenhuma *tag* padrão, o que fornece a habilidade de criar os próprios elementos de marcação e atributos que satisfaçam à sua aplicação, em outras palavras, criar a própria linguagem de marcação de acordo com a necessidade.

Em [FORNARI, 2003] há uma citação de algumas regras derivadas do XML que o XHTML adota sendo importante salientar algumas como, por exemplo:

- Existência de um único elemento raiz e este não pode fazer parte do conteúdo de nenhum outro elemento;

- Todas as *tags* devem ter abertura (ex: <*tag*>) e fechamento (ex: </*tag*>). Caso seja uma *tag* sem conteúdo ou nula o fechamento é indicado com uma barra inclinada "/" após o seu nome (ex: <*tag*/>);

- Os elementos das delimitados pela marcação inicial e final devem estar aninhados, ou seja, dentro de um mesmo elemento, não causando sobreposição de *tags*.

A linguagem XHTML foi criada com o intuito de ser um código HTML mais limpo e melhor formatado e a qual se atribui a característica de ser extensível, ou seja, no XHTML é permitido ampliar e modificar suas regras de sintaxe. Isto visando à melhor acessibilidade e compatibilidade de exibição de páginas Web em tipos de dispositivos diferenciados, como por exemplo, televisões, palmtops, celulares, smartfones, tablets, etc. Essa linguagem, diferentemente do HTML consegue ser interpretada por qualquer dispositivo, independente da plataforma utilizada, desde que exista suporte, isto porque as marcações possuem um sentido semântico para as máquinas.

Fazendo um histórico do XHTML tem-se que no ano de 2000 foram publicadas as primeiras especificações da linguagem, o XHTML 1.0. Basicamente ela é uma evolução da linguagem HTML 4.01, que foi publicada um ano antes, com a incorporação de características de documentos XML tornando a linguagem mais robusta e estrita. Um ano depois foram publicadas as especificações do XHTML 1.1, com

poucas alterações. Já em 2002 foi lançado o primeiro rascunho, do inglês, “draft” como é conhecido, do XHTML 2.0, uma nova versão do XHTML que faria um ruptura com o passado tirando a exigência de compatibilidade com as versões anteriores fazendo algumas alterações, priorizando a estruturação do código, e incluindo novos recursos, como XForms, XFrames, *tag* de listas de navegação, etc. Foram feitos oito rascunhos, sendo o último em 2008.

2.3 REFATORAÇÃO

O conceito de refatoração vem originalmente da comunidade de programação orientada a objetos aproximadamente no início da década de 90, mas foi popularizada em 1999 com Martin Fowler com o livro *“Refactoring”* (Adisson – Wesley, 1999).

Segundo [FOWLER, 1999], a refatoração é um processo de alteração de código-fonte de um sistema de software de modo que o comportamento observável não mude, mas que sua estrutura interna seja aperfeiçoada. Outro conceito, abordado por [Harold, 2010] diz que refatoração é a melhoria gradual de uma base de código por meio de pequenas mudanças que não modificam o comportamento de um programa. Isto normalmente é realizado com alguma ferramenta automatizada. A refatoração tem como objetivo, remover problemas de código – muitas vezes legado e produzir código mais claro e que seja mais fácil de manter, depurar e adicionar funcionalidades.

Baseados nestes conceitos pode-se dizer que a refatoração é uma forma disciplinada de realizar uma reestruturação do código utilizando mudanças pequenas para melhorar o projeto interno, visando promover atributos não funcionais de software, tais como legibilidade, manutenibilidade, complexidade, ou para reorganizar a arquitetura interna visando maior extensibilidade, sem mudar a semântica do mesmo.

Tecnicamente, a refatoração não corrige erros, não adiciona nem remove funcionalidades de um sistema.

2.4 REFATORAÇÃO PARA PÁGINAS WEB

Os conceitos e técnicas de refatoração são amplamente utilizados atualmente em linguagens de programação orientada a objetos, como por exemplo, Java e C#. Entretanto não é apenas código orientado a objetos e linguagens orientadas a objetos que pode produzir código ruim, no qual haveria a necessidade de refatoração. Indo mais além, não apenas as linguagens de programação. Praticamente todo o sistema

suficientemente complexo desenvolvido e mantido ao longo tempo pode se beneficiar de técnicas de refatoração.

Num sistema Web, no lado do servidor tem-se uma grande aplicação distribuída que na grande maioria das vezes funciona sobre uma base de dados relacional. Já no lado do cliente é uma ou mais páginas Web, normalmente desenvolvida(s) utilizando código HTML, estilos CSS e Javascript, que, dependendo de como e quando foram desenvolvidas, podem apresentar algumas dificuldades no momento em que o programador for implementar alguma nova funcionalidade ou simplesmente quer fazer alguma alteração. Por isto a importância da utilização de técnicas refatoração nesta área.

Na obra de Harold, [HAROLD, 2010], são abordadas mais de 60 sugestões de refatoração relacionados a códigos HTML subdivididas pelos assuntos *Documentos bem formados, Validade, Leiaute, Acessibilidade, Aplicações Web* e *Conteúdo*. Destas refatorações, neste trabalho foram selecionadas 10 que serão citadas a seguir:

- *Validade:*

(i) Substituir elemento I por EM ou CSS;

(ii) Substituir elemento B por STRONG ou CSS;

- *Leiaute:*

(iii) Substituir leiautes baseados em tabelas;

- *Acessibilidade:*

(iv) Adicionar rótulos para entrada de dados;

(v) Ligar preenchimento automático;

- *Segurança:*

(vi) Substituir GET inseguro por POST;

(vii) Substituir POST por GET;

(viii) Adicionar tipos de formulários Web 2.0;

(ix) Substituir Flash por HTML e

(x) Substituir formulários de contato por links MAILTO.

2.5 LINGUAGENS DE CONSULTA PARA BUSCA EM DOCUMENTOS COM LINGUAGENS DE MARCAÇÃO BEM FORMADOS

Nesta seção serão abordadas algumas linguagens de consulta utilizadas para busca em documentos bem formados (como o XML e o XHTML). Primeiramente será abordada a linguagem XPath que dentre as que serão descritas é a mais antiga. Após será descrita a linguagem XQuery que é considerada uma evolução da linguagem XPath, aproveitando-se de similaridades com a linguagem SQL.

2.5.1 XPath

XPath (*XML Path Language*) é uma linguagem de consulta para selecionar nós de um documento XML. Além disso, o XPath pode ser usado para computar valores(por exemplo strings, números ou valores booleanos) a partir do conteúdo de um documento XML. Esta linguagem foi definida pelo World Wide Web Consortium (W3C). [W3C XPath, 2011]

A partir deste conceito pode-se afirmar que o XPath é uma linguagem que trabalha com endereçamento de partes de documentos XML usando expressões de caminho similares a expressões de caminhos de diretórios em sistemas de arquivos em computadores.

Por exemplo: a partir do código XML do quadro 2.1 abaixo, pode-se definir algumas linhas de código em XPath.

/listadecompras : seleciona o nó raiz deste xml.

/listadecompras/item ou //item: seleciona todos os itens relacionados ao nó “listadecompras”.

/listadecompras/item[@id = “2”] : seleciona todos os itens relacionados ao nó “listadecompras” que contém um atributo chamado id com o valor igual a 2.

Figura 2.1: exemplo de código XML referente a uma lista de compras.

```
1     <?xml version="1.0" encoding="ISO-8859-1"?>
2     <listadecompras>
3        <item id =”1”>
4           <nome>arroz </nome>
5           <quantidade>2 </quantidade>
6           <preco>3.0 </preco>
7        </item>
8        <item id =”2”>
9           <nome>ovos</nome>
10          <quantidade>12 </quantidade>
11          <preco>2.50 </preco>
12       </item>
13       <item id =”3”>
14          <nome>feijao </nome>
15          <quantidade>1 </quantidade>
16          <preco>2.0 </preco>
17       </item>
18    </listadecompras>
```

A linguagem XPath tem diversas outras aplicações em outras especificações XML como XSLT e XQuery, por exemplo. Também pode-se utilizá-la em linguagens de programação para a recuperação e/ou localização de partes específicas de documentos XML.

2.5.2 XQuery

XQuery é uma linguagem de consulta de dados em documentos XML, construída sobre expressões XPath, similar à linguagem de consulta SQL, que foi desenvolvida pelo projeto XML Query, do W3C com o intuito de buscar e extrair informações que estão em documentos XML.

Assim, seu papel principal é obter informações para bases de dados XML, o que inclui bancos de dados relacionais que armazenam dados XML, ou que apresentam a visão XML dos dados que possuem. Entre alguns exemplos de uso para o XQuery estão escolha e transformação de dados XML para publicação Web e geração de relatórios de dados em banco de dados XML.

XQuery 1.0 e XPath 2.0 compartilham o mesmo modelo de dados e suporte as mesmas funções e operadores. [W3C XQuery, 2011].

A linguagem tem como expressão base de comandos que trabalham com iteração e ligação de variáveis para consultas XQuery. Esta é conhecida como FLWOR (que possui a pronuncia similar a “flower”). Esta expressão é um acrônimo para os comandos, “for”, “let”, “where”, “order by” e “return”. O comando “for” realiza a

iteração dos elementos, “let” faz a definição determinada variável, “where” define restrições as buscas, “order by” ordena os resultados e “return” mostra o resultado da consulta.

Para exemplificar buscas em XQuery será apresentado na Figura 2.2 o código referente ao documento XML *agenda.xml* que simula uma agenda que armazena dados de pessoas físicas e jurídicas.

Figura 2.2: exemplo de documento XML referente a uma agenda.

```
1   <?xml version="1.0" encoding="iso-8859-1"?>
2   <agenda>
3       <cadastrado id="1" tipo="pfisica">
4               <nome>Pablo Lima Flores</nome>
5               <cpf>973425600-09</cpf>
6               <email>pablo@inf.ufsm.br</email>
7               <telefone>9585-1234</telefone>
8       </cadastrado >
9       <cadastrado id="2" tipo="pfisica">
10              <nome>Joao da Silva</nome>
11              <cpf>948235600-00</cpf>
12              <email>joaodasilva@email.com.br</email>
13              <telefone>8432-7240</telefone>
14      </cadastrado >
15      <cadastrado id="3" tipo="pjuridica">
16              <nome>Serve Bem Lanches</nome>
17              <cnpj>8273482600-8</cnpj>
18              <email>contato@xislanches.com.br>
19              <telefone>3026-1991</telefone>
20      </cadastrado >
21      <cadastrado id="4" tipo="pfisica">
22              <nome>Maria Clara Gomes</nome>
23              <cpf>825822060-18</cpf>
24              <email>mclaragomes@email.com.br>
25              <telefone>3025-1555</telefone>
26      </cadastrado >
27  </agenda>
```

A Figura 2.3 exibe um exemplo de uma consulta XQuery que é aplicada ao documento XML apresentado na Figura 2.2: Este código representa a busca dos nomes e os emails dos elementos *cadastrado* com o atributo *id* com valor menor que 3.

Figura 2.3: Exemplo da consulta XQuery aplicada ao documento *agenda.xml*.

```
for $cad in doc("agenda.xml")//cadastrado
where $cad[@id < 3]
return
<retorno>
        <nomecadastrado>
                { $cad/nome/text()}
        </nomecadastrado>
        <emailcadastrado>
                { $cad/email/text()}
        </emailcadastrado>
</retorno>
```

Esta consulta exibe como resultado dois elementos *retorno* com o nome e o e-mail cadastrados. A Figura 2.4 exibe o resultado da consulta.

Figura 2.4: resultado da consulta XQuery apresentada na Figura 2.3.

```
1   <retorno>
2      <nomecadastrado>Pablo Lima Flores</nomecadastrado>
3      <emailcadastrado>pablo@inf.ufsm.br</emailcadastrado>
4   </retorno>
5   <retorno>
6      <nomecadastrado>Joao da Silva</nomecadastrado>
7      <emailcadastrado>joaodasilva@email.com.br</emailcadastrado>
8   </retorno>
```

No exemplo seguinte, a Figura 2.5 exibe um exemplo de outra consulta XQuery que é aplicada ao mesmo documento XML apresentado da Figura 2.2: O código corresponde a busca dos nomes e os CPFs das pessoas, que são os elementos *cadastrado* com o atributo *tipo* igual a "*pfisica*" ordenados pelo nome da pessoa e o total de pessoas que aparecem no resultado.

Figura 2.5: Exemplo de consulta XQuery.

```
<agendaretorno>
{
      for $cad in doc("agenda.xml")//cadastrado
      where $cad[@tipo = 'pfisica'] order by $cad/nome
      return
            <pessoa>
                {$cad/nome, $cad/cpf}
            </pessoa>
}
{
      let $cadastrado := doc("agenda.xml")//cadastrado
      return
             <totaldepessoas>
                {count($cadastrado[@tipo = 'pfisica'])}
             </totaldepessoas>
}
</agendaretorno>
```

Como resultado tem-se a criação de um elemento *agendaretorno* com 3 elementos *pessoa* com nome e CPF . A Figura 2.6 exibe o resultado desta consulta.

Figura 2.6: resultado da consulta XQuery.

```
1   <agendaretorno>
2     <pessoa>
3          <nome>Joao da Silva</nome>
4          <cpf>948235600-00</cpf>
5     </pessoa>
6     <pessoa>
7          <nome>Maria Clara Gomes</nome>
8          <cpf>825822060-18</cpf>
9     </pessoa>
10     <pessoa>
11          <nome>Pablo Lima Flores</nome>
12          <cpf>973425600-09</cpf>
13     </pessoa>
14     <totaldepessoas>3</totaldepessoas>
15  </agendaretorno>
```

É interessante perceber que em ambos os exemplos nota-se que na saída, no resultado das buscas há criação ou modificação das *tags*, recurso não compartilhado com o XPath, por exemplo. Este é um dos motivos pelo qual se adotou a escolha da utilização de XQuery neste trabalho.

3. IDENTIFICAÇÃO DE OPORTUNIDADES DE REFATORAÇÃO EM PÁGINAS XHTML

Neste capítulo serão abordadas algumas maneiras de busca de oportunidades de refatoração referentes a problemas recorrentes a más práticas de programação em documentos Web, especificamente documentos XHTML.

3.1 DESCRIÇÕES DO PROCESSO GERAL

Um processo de desenvolvimento de software é normalmente composto por um conjunto de atividades inter-relacionadas, associado a um conjunto de papéis, e ao requerido, artefatos produzidos ou modificados. Uma atividade expressa em um processo deve ocorrer de forma a criar uma peça de software. Exemplos deatividades são: criar casos de uso, analisar os riscos, o modelo de banco de dados e projetar a interface de usuário (KRUTCHEN, 2000; Coplien; HARRISON, 2005).

Um papel é uma função atribuída ou usurpada ou posição em um processo de desenvolvimento de software. Ela define as responsabilidades da pessoa envolvida no processo. Exemplos de papéis típicos em processos de desenvolvimento desoftware são: arquiteto de software, gerente de projetos, designer de interface do usuário, desenvolvedor, analista, testador, e escritores técnicos [KRUTCHEN, 2000].

Artefatos são todos os documentos necessários ou gerados durante a realização das tarefas descritas em cada atividade. Por exemplo, documentos normalmente manipulados em um processo de desenvolvimento de software são: modelos de banco de dados, código-fonte, documentos de requisitos e desenho de software de arquitetura. Elementos adicionais em um processo de incluir diretrizes, os mentores de ferramentas e roteiros [KRUTCHEN, 2000].

Atividades relacionadas são agrupadas em disciplinas (também chamados de fluxos de trabalho) [KRUTCHEN, 2000]. A disciplina é um conjunto organizado de atividades inter-relacionadas que encapsula uma preocupação central do processo.

Uma disciplina contendo atividades de refatoração pode ajudar os desenvolvedores a organizar o processo de refatoração, a fim de minimizar os esforços necessários e maximizar os resultados efetivos. Isto é conseguido através da melhoria de um módulo de software específico em um conjunto de atributos de qualidade escolhido,

e pela aplicação de um conjunto de padrões de refatoração selecionados que contribuam para a melhoria dos atributos de qualidade selecionados em um conjunto de elementos selecionados - escolhido por suas chances de ser melhorada pelos padrões selecionados. [PIVETA, 2009]

Esta seção tem como o objetivo descrever uma disciplina para refatoração de software para Web e definir o escopo deste trabalho. É composto de um conjunto de atividades do núcleo, um conjunto de papéis, e um conjunto de artefatos necessários para identificar, priorizar oportunidades de refatoração, e aplicar padrões de refatoração em aplicações de software.

A Figura 3.1 mostra estas atividades utilizando um diagrama [PIVETA, 2009]. Cada atividade será explicada a seguir.

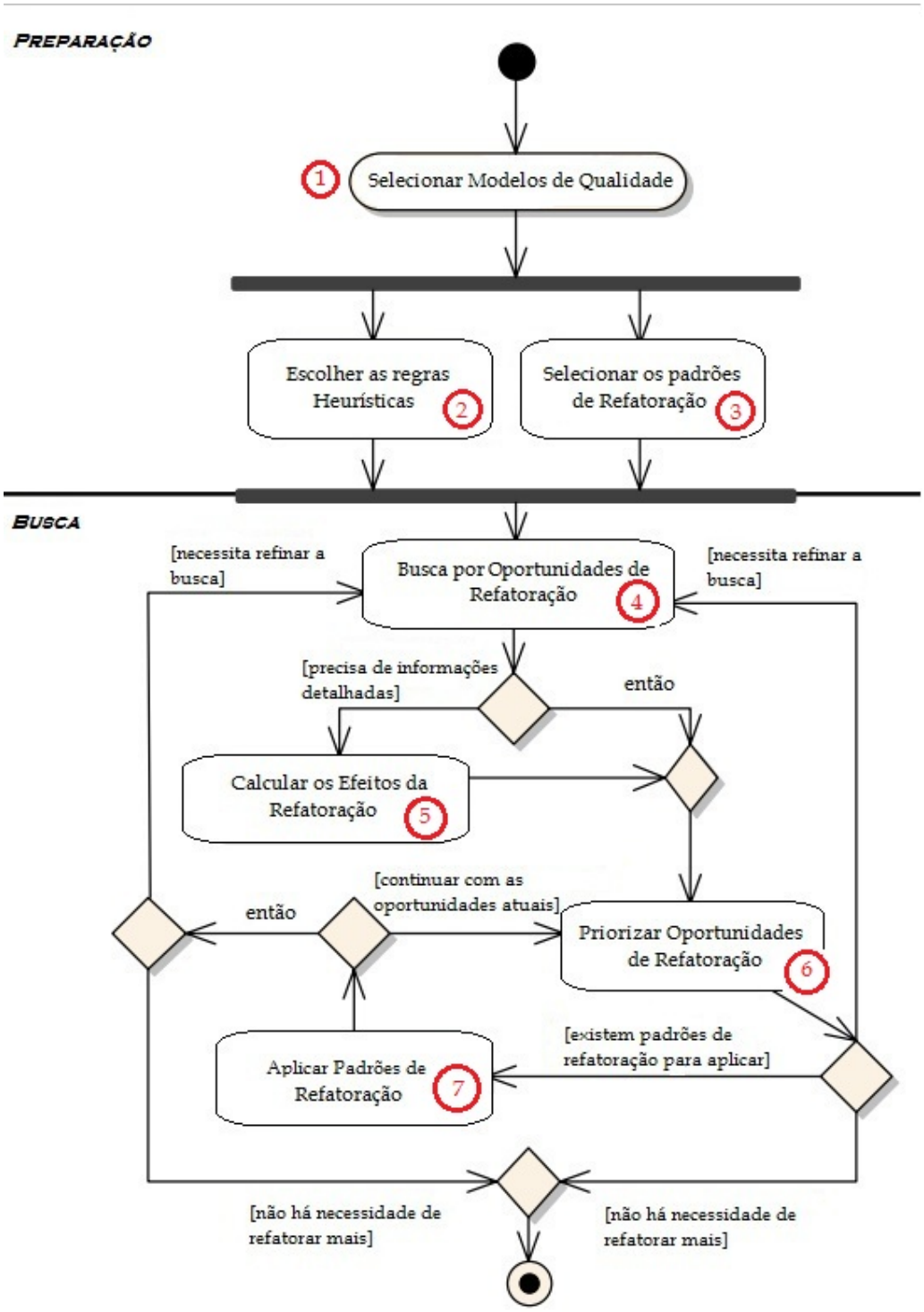

Figura 3.1: Diagrama de atividades de relativas a refatoração.

Segundo [PIVETA, 2009], as atividades são descritas resumidamente a seguir:

1 - Selecionar ou Criar Modelos de Qualidade: O primeiro passo é selecionar um modelo de qualidade para que o aplicativo que será avaliado. Este foco de atividade no reconhecimento de quais atributos de qualidade que serão considerados no processo de refatoração. Além disso, há a necessidade de definir que métricas são associados a esses atributos de qualidade, para permitir uma avaliação qualitativa da qualidade de software em termos de modelo de qualidade selecionado e/ou criado. Exemplos de atributos de qualidade são: reutilização, compreensibilidade, legibilidade, confiabilidade. Exemplos de métricas são: linhas de código, número de operações no módulo e profundidade da árvore de herança.

2 - Escolher ou criar regras heurísticas: O próximo passo inclui a seleção ou criação de funções quantitativas relacionadas a atributos de qualidade selecionados e métricas, chamada regras heurísticas. Estas regras heurísticas foco em expressar a relação entre modelos de qualidade, deficiências e elementos de software de uma forma quantitativa. Mesmo embora eles não garantem melhores resultados, eles podem fornecer uma boa estimativa de as qualidades que os desenvolvedores esperam que a partir da aplicação de software.

3 - Selecionar Padrões de refatoração: Paralelo à seleção e/ou criação de regras heurísticas, os desenvolvedores devem selecionar um conjunto de padrões de refatoração de catálogos de padrões de refatoração e ordená-los de acordo com seu impacto estimado sobre o modelo de qualidade selecionada, criando um ranking de padrões de refatoração. O foco está na seleção dos padrões de refatoração que são mais susceptíveis de melhorar os atributos de qualidade de software. O desenvolvedor pode definir limites no ranking de tais padrões de refatoração para aumentar ou diminuir o número de padrões refatoração avaliados.

4 - Buscar de oportunidades de refatoração: Dado um conjunto de padrões de refatoração, um modelo de qualidade, um conjunto de regras heurísticas, e um conjunto de elementos de software selecionado (um pacote, um conjunto de classes, uma classe única), os desenvolvedores podem procurar o aplicativo de software para possíveis oportunidades para aplicar esses padrões de refatoração. A idéia é melhorar a qualidade do software, concentrando-se nas oportunidades que são mais susceptível de produzir efeitos positivos em um conjunto de atributos de qualidade selecionada.

5 - Calcular os Efeitos da Refatoração: Depois que as oportunidades refatoração foram identificadas, os desenvolvedores podem avaliar os efeitos da

refatoração selecionados padrões em artefatos de software, de acordo com o modelo de qualidade selecionada. Nesta atividade, o foco está em uma avaliação quantitativa. O uso de funções de impacto é proposto como uma técnica para permitir a desenvolvedores quantitativamente calcular os efeitos da refatoração. Outra forma de calcular os efeitos da refatoração é aplicar os padrões de refatoração e avaliar os resultados.

6 - Priorizar Oportunidades de Refatoração: Depois que as oportunidades refatoração são identificadas e os efeitos de cada um são computados, o desenvolvedor avalia de forma quantitativa as alterações propostas e decidem quais são vantajosos. Nesta atividade, o desenvolvedor deve ser capaz de filtrar e ordenar as oportunidades refatoração e marcá-los conforme o caso ou não. O desenvolvedor pode então passar para a próxima atividade: aplicar padrões de refatoração.

7 - Aplicar Padrões de Refatoração: A próxima atividade é a aplicação de padrões de refatoração para os elementos de software e re-executar os casos de teste disponíveis para garantir que a aplicação padrões de refatoração não prejudique nada. Os resultados podem ser analisados e, se necessário, os efeitos dos padrões de refatoração pode ser desfeita. O desenvolvedor pode reiniciar a pesquisa, refinando os critérios de pesquisa, filtros, ordenação e começar de novo a análise das oportunidades refatoração sugerida. Para tornar mais profunda as mudanças no modelo de qualidade, nas regras heurísticas ou no refatoração suportados padrões, o desenvolvedor pode reiniciar o processo, desde o início.

Destas atividades abordadas no diagrama, será abordada de maneira mais detalhada na seção seguinte a atividade relacionada à busca de oportunidades de refatoração, no qual é focado o tema deste trabalho.

3.2 BUSCA DE OPORTUNIDADES DE REFATORAÇÃO

Em aplicações de software, diversas oportunidades para a aplicação de padrões de refatoração podem ser encontradas. Como visto na seção 3.1, um desenvolvimento de software pode ser visto como conjunto de atividades inter-relacionadas, associado a um conjunto de papéis, e ao requerido, artefatos produzidos ou modificados. Em relação à busca de oportunidades para refatoração podemos definir como um conjunto, apresentado na Figura 3.2, formado um papel, uma atividade e três artefatos.

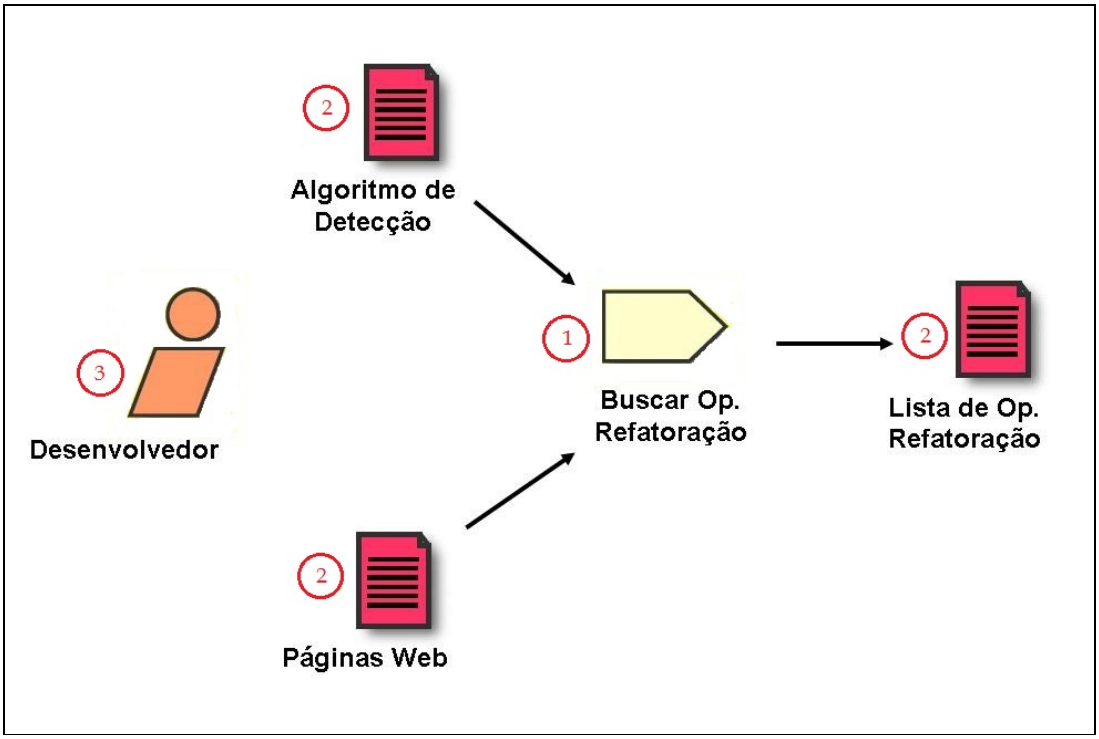

Figura 3.2: Conjunto de atividades, artefatos e papel relacionado à busca de oportunidades de refatoração.

1 - Atividade:

a) Buscar Oportunidades de Refatoração: é a atividade de busca de oportunidades de refatoração pode ser realizada pelos desenvolvedores com a ajuda de ferramentas automatizadas.

2 - Artefatos:

a) Algoritmo de Detecção: é o algoritmo que contém a idéia da refatoração a ser buscada, de acordo com o padrão escolhido.

b) Pagina Web: é a página em XHTML que esta sendo avaliada de acordo com o escopo da pesquisa pelas oportunidades de refatoração.

c) Lista de Oportunidades de Refatoração: é criada depois que a pesquisa é realizada. Esta lista é avaliada para decidir qual dos padrões de refatoração associados com as oportunidades de refatoração será aplicado.

3 - Papel:

a) Desenvolvedor: aquele que desenvolve os algoritmos e cria ferramentas e programas para a realização das buscas.

3.3 CATÁLOGO - PROBLEMAS COMUNS DE CRIAÇÃO DE PÁGINAS XHTML

Nesta seção são listados alguns problemas que relacionado ao desenvolvimento de páginas Web utilizando XHTML. Os problemas foram subdivididos de acordo com o padrão de refatoração desejado em:

- Problemas de Validade;

- Problemas de Leiaute;

- Problemas de Acessibilidade e

- Problemas de segurança e desempenho em aplicações Web.

Para a criação de exemplos de documentos XML, desenvolvimento das buscas XQuery e realização de testes foi utilizado a IDE XMLSpy da Altova [ALTOVA XMLSpy]. O XMLSpy é um editor XML para modelagem, edição, transformação e depuração em tecnologias relacionados com XML trabalhando com DTDs, esquemas, XSL, XPath, XQuery entre outros.

3.3.1 Problemas de Validade

Nesta seção serão abordados alguns problemas referentes à validade de documentos. Em questão do quesito qualidade, as refatorações visam garantir que os documentos sejam bem formados e tenham corretude sintática e semântica.

3.3.1.1 Refatoração 1: Substituição do elemento *b* por strong ou utilização de CSS

Esta operação consiste em sugerir a modificação do uso de tags *b*(exemplificado na figura 3.3), caso existam, para tags *strong*(com um significado semântico mais adequado), dependendo do tipo de ênfase que se deseja apresentar também se pode substituir o a ocorrência dos elementos *b* por algum estilo CSS.

Figura 3.3: Exemplo de trecho de código que possui alguns caracteres em negrito utilizando a tag *b*.

```
1    <p>A linguagem HTML, que é abreviação de <b>H</b>yper<b>t</b>ext
2    <b>M</b>arkup <b>L</b>anguage - Linguagem de Marcação de HiperTexto,
3        é uma linguagem de marcação que segue os padrões da <b>W3C</b>
4        voltada principalmente para criação de páginas Web.</p>
```

O resultado dessa possível refatoração é apresentado na Figura 3.4 com a substituição de todas as tags *b* para tags *strong*.

Figura 3.4: Refere-se ao trecho de código apresentado na figura 3.3 substituindo-se todos os elementos ou tags *b* por elementos *strong*.

```
1  <p>A linguagem HTML,que é abreviação de <strong>H</strong>yper<strong>t</strong>ext
2  <strong>M</strong>arkup <strong>L</strong>anguage - Linguagem de Marcação de
3   HiperTexto, é uma linguagem de marcação que segue os padrões da
4   <strong>W3C</strong> voltada principalmente para criação de páginas Web.</p>
```

- Como detectar:

Uma forma de verificar esta possibilidade de refatoração é buscar todas as ocorrências do elemento *b* no documento.

- Implementação da detecção:

Figura 3.5: Código em XQuery que retorna todos os elementos *b* do documento.

```
let $b := doc("index.xhtml")//b
return
$b
```

3.3.1.2 Refatoração 2: Substituição do elemento *i* por *em* ou utilização de CSS

Similar ao item anterior, a operação consiste em sugerir a modificação do uso de tags *i*, caso existam para tags *em*, se a utilização da tag *i* foi puramente para dar ênfase, ou substituir por elementos CSS.

Figura 3.6: Exemplo de trecho de código que possui alguns caracteres em itálico utilizando a tag *i*.

```
1          <p class="content">Lixo eletrônico (termo que não deve ser
2          confundido com <i>spam</i>), é o nome dado aos resíduos
3          resultantes da rápida obsolescência de equipamentos
4          eletrônicos (o que inclui televisores, celulares, computadores,
5          geladeiras e outros dispositivos). </p>
```

O resultado da refatoração aplicada no código da Figura 3.6 é apresentado na Figura 3.7 com a substituição da tag *i* para tag *em*.

Figura 3.7: Código similar ao da Figura 3.6 com a modificação da tag *e* para a tag *em*.

```
1        <p class="content">Lixo eletrônico (termo que não deve ser
2        confundido com <em>spam</em>), é o nome dado aos resíduos
3        resultantes da rápida obsolescência de equipamentos
4        eletrônicos (o que inclui televisores, celulares, computadores,
5        geladeiras e outros dispositivos). </p>
```

- Como detectar:

Analogamente ao item anterior, uma forma de verificar estas possibilidades de refatoração é buscar todas as ocorrências do elemento *i* no documento.

- Implementação da detecção:

Figura 3.8: Código em XQuery que retorna todos os elementos *i* do documento.

```
let $i := doc("index.xhtml")//i
return
$i
```

3.3.2 Problemas de Leiaute

Nesta seção serão abordados problemas referentes ao leiaute. A partir do momento que o documento é bem formado sintaticamente, o próximo passo é verificar se as restrições semânticas são apropriadas. Usar a semântica apropriada para cada elemento torna as páginas mais inteligíveis para os leitores de tela e faz com que elas sejam apresentadas apropriadamente em diversas plataformas. A seguir serão mostrados alguns exemplos a serem trabalhados.

3.3.2.1 Refatoração 3: Substituição de leiautes baseados em tabelas

Atualmente tem-se trabalhado bastante com o desenvolvimento do leiaute de páginas Web utilizando o conceito “tableless”, ou seja, sem tabelas. Para a criação do leiaute a W3C recomenda a utilização de CSS restringindo a utilização das tabelas apenas para apresentação de dados tabulares. Este padrão de desenvolvimento facilita a separação da camada de apresentação para em arquivo CSS, possibilitando a manutenção da página mais facilmente, entre outras vantagens.

A operação de busca consiste em verificar o uso de leiautes de páginas Web que são baseados em tabelas, exemplificado pelo código da figura 3.9 e sugerir uma mudança desse leiaute para a utilização de tags *div* juntamente com a utilização de estilos CSS.

Figura 3.9: Exemplo de página com seu leiaute construído com o uso de elementos *table*.

```
1     <html xmls = “http://WWW.w3c.org/1999/xhtml”>
2     <head>
3          <title>Pagina com leiaute de 3 colunas</title>
4     </head>
5     <body>
6          <h3>Exemplo de página estruturada em 3 colunas:</h3>
7          <table>
8               <tr>
9                    <td valign=”top” id=”esquerda”>Coluna da esquerda</td>
10                   <td valign=”top” id= “centro”>Coluna central</td>
11                   <td valign=”top” id=”direira”>Coluna da direita</td>
12              </tr>
13         </table>
14     </body>
15     </html>
```

Uma possível refatoração aplicada ao código da Figura 3.9 para a substituição do elemento *table* por elementos *div* com o acréscimo de estilos CSS.

Figura 3.10: Página com leiaute construído com o uso de elementos *div* e CSS.

```
1     <html xmls = “http://WWW.w3c.org/1999/xhtml”>
2     <head>
3          <title>Pagina com leiaute de 3 colunas</title>
4          <link rel=”stylesheet” href=”trescolunas.css” type=”text/css” />
5     </head>
6     <body>
7          <h3>Exemplo de página estruturada em 3 colunas:</h3>
8          <div id=”esquerda”>Coluna da esquerda</div>
9          <div id=”centro”>Coluna central</div>
10         <div id=”direita”>Coluna da direita</div>
11    </body>
12    </html>
```

- Como detectar:

Para detectar esta situação deve-se procurar por ocorrências da tag *table* e verificar se esta define parte do leiaute da página ou apenas uma tabela utilizada na página.

- Implementação da detecção:

Figura 3.11: Código em XQuery que retorna todos os elementos *table* para avaliação.

```
let $table := doc("index.xhtml")//table
return
$table
```

3.3.3 Problemas de Acessibilidade

A preocupação com acessibilidade no desenvolvimento de sistemas Web é crescente. Acessibilidade trabalha entre outras coisas com a disseminação da informação para um publico maior de usuários, que às vezes algum tipo de problema físico de visão, atenção, problemas motores, etc.

Páginas Web bem projetadas independem se o usuário está lendo-as utilizando um monitor de vídeo ou um leitor de tela. Formulários bem projetados também independem se o usuário está informando os dados por um teclado, mouse ou um aplicativo de software de reconhecimento de voz.

Nesta seção serão abordadas algumas situações que podem ser analisadas e/ou modificadas de acordo com o objetivo da aplicação.

3.3.3.1 Refatoração 4: Adicionar rótulos à entrada de dados

Usuários com limitações visuais que utilizam leitores de telas nem sempre podem usar o leiaute bidimensional de uma página para determinar quais rótulos são associadas as quais campos. É importante rotular explicitamente cada um dos campos não ocultos, de forma que cada um deles seja corretamente identificado por leitores de telas. Da maneira como está apresentado o código da Figura 3.12, leitores de tela podem não apresentar os rótulos de maneira adequada.

Figura 3.12: Trecho de código de um formulário cujos rótulos não estão envolvidos por elementos *label*.

```
1   <form action="login.php" method="post">
2       <p>
3               Usuário:
4               <input type="text" name="log" id="log" size="18"/>
5       </p>
6       <p>
7               Senha:
8               <input type="senha" name="sen" id="sen" size="18"/>
9       </p>
10      <p>
11              <input name="stayloggein" type="checkbox" id="stayloggein" value="remember"/>
12              Lembrar-me neste computador
13      </p>
14      <p class="submit">
15              <input type="submit" name="submit" id="submit" value="login"/>
16              <input type="hidden" name="redirect" value="admin.php"/>
17      </p>
18   </form>
```

Adicionar elementos *label* explícitos para rótulos juntamente com os atributos *id* e *class* facilitará a aplicação de estilos CSS ou fornecendo dicas adicionais utilizando navegadores inteligentes. A Figura 3.13 mostra o acréscimo do elemento *tag* nos rótulos dos campos do formulário.

Figura 3.13: Representa a *refatoração 4* aplicada ao formulário da Figura 3.12

```
1   <form action="login.php" method="post">
2       <p>
3          <label for="log">Usuário:
4             <input type="text" name="log" id="log" size="18"/>
5          </label>
6       </p>
7       <p>
8          <label for="sen">Senha:
9             <input type="senha" name="sen" id="sen" size="18"/>
10         </label>
11      </p>
12      <p>
13         <label for="stayloggein">
14            <input name="stayloggein" type="checkbox" id="stayloggein" value="remember"/>
15            Lembrar-me neste computador
16         </label>
17      </p>
18      <p class="submit">
19         <input type="submit" name="submit" id="submit" value="login"/>
20      </p>
21   </form>
```

- Como detectar:

Para detectar esta situação será verificado, para cada tag *form*, todas as ocorrências da tag *input* que não sejam do tipo *submit* cujos pais são diferentes de *label*.

- Implementação da detecção:

Figura 3.14: Consulta XQuery que verifica a existência de tags *input* sem o nó pai *label*.

```
for $form in  doc("refatoracao3.2.3.1-rotulos-de.xml")//form
let $input := $form//input[@type != "submit"]
return $input/parent::*[name() != "label"]
```

3.3.3.2 Refatoração 5: Ligar preenchimento automático de formulários

Esta é uma questão que gera certa discussão em relação à segurança da informação principalmente em computadores públicos. Mas pode-se afirmar que a utilização do preenchimento automático evita que os usuários percam tempo digitando conteúdo repetido. Em especial, usuário com algum tipo de limitação física, devido a doenças ou idade. Esta decisão cabe a equipe de desenvolvimento e depende basicamente do tipo de aplicação Web desenvolvido, quais serão os seus usuários e que nível de segurança é exigido. O código da Figura 3.15 demonstra a opção de utilizar o atributo *autocomplete = "off"* , ou seja, desliga a função de preenchimento automático.

Figura 3.15: Código de um formulário de preenchimento de e-mail no qual os seus campos de não possuem de preenchimento automático.

```
1   <form action="/login" method="post" autocomplete="off">
2      <p>
3         <label>E-mail:
4            <input type="text" name="e1" autocomplete="off"/>
5         </label>
6      </p>
7      <p>
8         <label>Senha:
9            <input type="senha" name="p1"/>
10        </label>
11     </p>
12     <input type="submit" title="Login" autocomplete="off"/>
13  </form>
```

Se o critério for permitir o preenchimento automático aos campos dos formulários da aplicação, basta omitir o atributo *autocomplete* no elemento *form* ou atribuí-lo o valor *on*. A Figura 3.16 apresenta o mesmo trecho de código da figura 3.15 com a aplicação da *refatoração 5*.

Figura 3.16: Formulário de preenchimento de e-mail com preenchimento automático.

```
1   <form action="/login" method="post">
2      <p>
3         <label>E-mail:
4            <input type="text" name="e1"/>
5         </label>
6      </p>
7      <p>
8         <label>Senha:
9            <input type="senha" name="p1"/>
10        </label>
11     </p>
12     <input type="submit" title="Login"/>
13  </form>
```

- Como detectar:

Neste caso basta verificar todas as ocorrências do atributo *autocomplete* com o valor igual a *off* em formulários e/ou *inputs* na página, verificando se realmente é necessário desabilitar este recurso.

- Implementação da detecção;

Figura 3.17: Consulta XQuery que retorna todos elementos *input* e *form* que não utilizam autocompletar.

```
for $form in doc("refatoracao3.2.3.2-autocompletar-de.xml")//form
let $input := $form//input
return
$form[@autocomplete="off"] | $input[@autocomplete="off"]
```

3.3.4 Problemas de Desempenho e Segurança em aplicações Web

Nesta seção serão abordados alguns problemas relacionados a desempenho e segurança em aplicações Web.

3.3.4.1 Refatoração 6: Substituir requisições GET por POST

Métodos ou requisições GET utilizam a própria URL (Uniform Resource Locator) para enviar dados ao servidor, quando se envia um formulário pelo método GET, o navegador verifica as informações do formulário e adiciona a URL de onde o formulário será enviado e o envia, separando o endereço da URL dos dados do formulário por um "?" (ponto de interrogação).

A busca de oportunidades em relação à refatoração 6 consiste em sugerir a recodificação de operações inseguras com envio via GET para que sejam enviadas via POST. Alguns procedimentos, como, por exemplo, confirmação de inscrição, fazer um pedido, concordar com um contrato, apagar uma página, inserir conteúdo numa base de dados, inserir comentário em um blog devem ser realizados, se possível, sempre via POST para evitar que estas ações possam ser realizadas acidentalmente sem o consentimento ou pedido do usuário. O código da Figura 3.18 abaixo mostra um exemplo de requisição GET que, para melhor segurança da informação, deveria ser codificada utilizando um formulário POST.

Figura 3.18: Código de um link que utiliza uma requisição GET para realizar uma ação de apagar.

```
1  <a class="delete"
2      href="texto.php?actiondelete&amp;id=1000517&amp;nonce=76a62"
3      onclick="return delete('post', 1000517, 'voce esta prestes a deletar este post &quot;
POST vs GET&quot;.\n&quot;;OK&quot; para deletar, &quot;Cancel&quot; para 4
cancelar');">Deletar</a>
```

Para realização dessa mesma ação recomenda-se o uso de requisições POST, normalmente implementada por formulários. A Figura 3.19 demonstra como deveria ser feito o procedimento de apagar.

Figura 3.19: Código de um link que utiliza uma requisição POST para realizar uma ação de apagar.

```
1  <form method="post" action="texto.php">
2      <input name="action" value="deletar" type="hidden" />
3      <input name="id" value="1000517" type="hidden" />
4      <input name="nonce" value="76a62" type="hidden" />
5      <input name="submit" value="Deletar" />
6  </form>
```

- Como detectar:

Como as requisições GET podem ser realizadas de 2 maneiras é interessante que haja 2 tipos de buscas diferentes para encontrá-las:

1) “Se a requisição foi feita a partir de uma URL: uma forma de encontrar URL que utilizem GET é primeiro, encontrar um link (tag a com atributo href) e verificar se no valor do atributo href existe algum caractere ‘?’. O que vier após este caractere podem ser parâmetros da requisição GET.
2) Se a requisição foi feita a partir de um formulário: basta buscar todos os formulários que possuem o atributo *method* com o valor *get*.

- Implementação da detecção:

Figura 3.20.1: Busca de oportunidades em requisições GET codificado através de URL.

```
1   let $doc := doc("refatoracao3.2.4.1-get-post-de.xml")
2   return $doc//*[contains(@href, "?")]
```

Caso o método de envio da requisição seja um formulário tem-se que fazer uma consulta diferente.

Figura 3.20.2: Busca de oportunidades em requisições GET codificado através de formulário.

```
1   for $form in doc("refatoracao3.2.4.1-get-post-de.xml")//form
2   return $form[@method="get"]
```

3.3.4.2 Refatoração 7: Substituir requisições POST por GET

As requisições ou métodos POST enviam os dados colocando-os no corpo da mensagem. Ele deixa a URL separada dos dados que serão enviados e com isso pode-se enviar qualquer tipo de dados por este método.

A idéia básica da refatoração 7 consiste em reprojetar as operações seguras que estão implementadas via POST para GET. Segundo [Harold, 2010], URLs que podem ser acessadas por GET podem ser referenciadas por links, marcadas como itens favoritos, precarregadas, armazenadas em cache, permite o uso de um botão ”voltar”, entre outros.

Alguns exemplos de operações que podem ser feitas via GET, pois são seguras e não trazem obrigações para o leitor são: ler e-mail, visualizar mapa, inspecionar estado corrente de uma máquina, ler documento, etc.

A figura 3.21 apresenta um exemplo de formulário de uma lista de itens que utiliza o método POST. Caso seja feita a refatoração 6 neste exemplo, o código seria equivalente a Figura 3.22.

Figura 3.21: Formulário utilizando método POST de uma lista de itens que contém nome e preço.

```
1   <form method="post" action="listaitens.php">
2      <p>
3         <label>Exibir    <input name="numero" value="10" type="text" /> itens      </label>
4      </p>
5      <p>
6         <label>
7            Organizados pelo:
8            <select name="sort">
9               <option value="nome"> Nome </option>
10              <option value="preco"> Preço </option>
11              </select>
12        </label>
13     </p>
14     <p><input type="submit" value="Lista de itens"/></p>
15  </form>
```

Figura 3.22: Formulário utilizando método GET de uma lista de itens que contém nome e preço.

```
1   <form method="get" action="listaitens.php">
2      <p>
3         <label>Exibir    <input name="numero" value="10" type="text" /> itens      </label>
4      </p>
5      <p>
6         <label>
7            Organizados pelo:
8            <select name="sort">
9               <option value="nome"> Nome </option>
10              <option value="preco"> Preço </option>
11              </select>
12        </label>
13     </p>
14     <p><input type="submit" value="Lista de itens"/></p>
15  </form>
```

- Como detectar:

Normalmente as requisições POST são realizadas através de formulários então uma maneira eficiente de realizar a pesquisa é verificar todas as ocorrências de formulários e comparar se o valor do atributo *method* é igual a *post*.

- Implementação da detecção:

Figura 3.23: Busca feita em XQuery que retorna todos os elementos *form* que possuem o atributo *type* com o valor *post*.

```
1   for $form in doc("refatoracao3.2.4.2-post-get-de.xml")//form
2   return $form[@method="post"]
```

3.3.4.3 Refatoração 8: Adicionar tipos de formulários Web 2.0

Segundo a especificação da WATHWG [WHATHWG WEB 2.0], os Formulários Web 2.0 é uma extensão das características dos formulários encontradas no HTML 4. Formulários Web 2.0 aplica-se tanto HTML e XHTML, e fornece novos campos de entrada fortemente tipados, novos atributos para restrições definição, um modelo de repetição para declarativa de seções do formulário, novas interfaces e eventos DOM, novos eventos DOM para validação e acompanhamento de dependência, XML entre outros.

Alguns exemplos dos novos tipos de dados que podem ser utilizados de acordo com que se espera de entrada no formulário: *email, date, time, datetime, localdatetime, month, week, number e url.* Estes tipos permitem que os navegadores forneçam componentes mais apropriados de interface gráfica com o usuário para a entrada de dados.

A figura 3.24 apresenta o código de um formulário cujos campos estão codificados de uma maneira genérica, sem definição exata dos tipos de dados que serão incluídos.

Figura 3.24: Formulário com campos genéricos.

```
1   <form action="http://www.paginadeexmplo.com.br/formulario" method="post">
2       <p><label>URL: <input name="url" /></label></p>
3       <p><label>Email:        <input name="email" /></label></p>
4       <p><label>Data:         <input name="data" /></label></p>
5       <p><label>Hora:         <input name="hora" /></label></p>
6       <p><label>Data e Hora:  <input name="dataehora" /></label></p>
7       <p><label>Data e Hora local:     <input name="dataehoralocal" /></label></p>
8       <p><label>Mês: <input name="mes" /></label></p>
9       <p><label>Semana:       <input name="semana" /></label></p>
10      <p><label>Número:       <input name="numero" /></label></p>
11      <p><label><input type="submit" value="Enviar" /></label></p>
12  </form>
```

Neste caso sugere-se a inclusão dos tipos específicos de dados que se quer trabalhar, evitando ambigüidades e possíveis erros de formatação na entrada dos dados nos formulários, como na figura 3.25.

Figura 3.25: Formulário web 2.0 exemplificando os vários tipos de dados que podem ser trabalhos.

```
1   <form action="http://www.paginadeexmplo.com.br/formulario" method="post">
2       <p><label>URL: <input type="url" name="url" /></label></p>
3       <p><label>Email:        <input type="email" name="email" /></label></p>
4       <p><label>Data:         <input type="date" name="data" /></label></p>
5       <p><label>Hora:         <input type="time" name="hora" /></label></p>
6       <p><label>Data e Hora:  <input type="datetime" name="dataehora" /></label></p>
7       <p><label>Data e Hora local:     <input type="localdatetime" name="dataehoralocal"
/></label></p>
8       <p><label>Mês: <input type="month" name="mes" /></label></p>
9       <p><label>Semana:       <input type="week" name="semana" /></label></p>
10      <p><label>Número:       <input type="number" name="numero" /></label></p>
11      <p><label><input type="submit" value="Enviar" /></label></p>
12  </form>
```

- Como detectar:

Neste caso basicamente todos os formulários são alvos em potencial. Então basta buscar todas as tags *form* que o arquivo possuir. Uma vez que um formulário for encontrado deve-se avaliar que tipo de dado é esperado para o campo esta sendo usado e verificar se não pode ser incluído/substituído um dos tipos citados anteriormente.

- Implementação da detecção:

Figura 3.26: Código referente a busca por elementos *input* que não possuam o tipo definido segundo formulário Web 2.0.

```
for $form in doc("refatoracao3.2.4.3-form2.0-de.xml")//form
let $input := $form//input

let $email := $input[@name="email" and (not(@type) or @type!="email")]

let $date := $input[(@name="data" or @name="date") and (not(@type) or
@type!="date")]

let $time := $input[(@name="hora" or @name="time") and (not(@type) or
@type!="time")]

let $datetime := $input[(@name="dataehora" or @name="datetime") and (not(@type) or
@type!="datetime")]

let $localdatetime := $input[(@name="dataehoralocal" or @name="localdatetime") and
(not(@type) or @type!="localdatetime")]

let $month := $input[(@name="mes" or @name="month") and (not(@type) or
@type!="month")]

let $week := $input[(@name="semana" or @name="week") and (not(@type) or
@type!="week")]

let $number := $input[(@name="numero" or @name="number") and (not(@type) or
@type!="number")]

let $url := $input[@name="url" and (not(@type) or @type!="url")]

return
<retorno>
{$email, $date, $time, $datetime, $localdatetime, $month, $week, $number, $url}
</retorno>
```

3.3.4.4 Refatoração 9: Substituir elementos Flash para HTML

Basicamente, a intenção dessa busca é sugerir algumas alternativas HTML (com ou sem o uso de CSS) ao desenvolvedor a mudança de algumas funcionalidades que por ventura foram projetadas com o uso de Flash, como por exemplo, o código fonte da Figura 3.27 que mostra uma página inicial de um site com uso de Flash.

Figura 3.27: Página inicial de um site cujos menus de navegação e o leiaute foram criados utilizando a tecnologia Flash.

```
1   <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
2       <head>
3               <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2"/>
4               <title>Meu site em flashl</title>
5       </head>
6<!--           Site desenvolvido em flash: o arquivo "index.swf" contém a apresentação da página com seus menus
direcionando aos conteúdos correspondentes.-->
7       <body bgcolor="#4d412f">
8               <div align="center">
9                       <object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=8,0,0,0"
width="775" height="680" id="index" align="middle">
10                              <param name="allowScriptAccess" value="sameDomain"/>
11                              <param name="movie" value="index.swf"/>
12                              <param name="quality" value="high"/>
13                              <param name="bgcolor" value="#4d412f"/>
14                              <embed src="index.swf" quality="high" bgcolor="#4d412f" width="775"
height="680" name="index" align="middle" allowScriptAccess="sameDomain" type="application/x-
shockwave-flash" pluginspage="http://www.macromedia.com/go/getflashplayer"/>
15                      </object>
16              </div>
17      </body>
18  </html>
```

Segundo Harold [Harold, 2010], existem tarefas que nunca devem ser escritas em aplicações Flash: Navegação, conteúdo de texto, imagens não iterativas slides, formulários de entrada de dados, faturas, etc. estas tarefas podem ser manipuladas com a utilização de XHTML puro com o suporte de estilos CSS, como mostra a Figura 3.28 e, ocasionalmente adicionar-se um pouco de Javascript.

Figura 3.28: Exemplo de página que não utiliza Flash.

```
1   <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
2       <head>
3               <title>Pagina com navegação em HTML</title>
4       </head>
5       <body>
6               <div id="conteudo">
7                       <h1>Página sem apresentação flash</h1>
8                       <p>Esta página foi desenvolvida sem o uso de recursos flash.</p>
9                       <p>Tarefas comuns como: navegação, slides, formulários de entrada de
dados, etc, de preferência não devem ser escritas em flash</p>
10              </div>
11              <div id="navega">
12                      <ul>
13                              <li><a href="item1/"> Item 1</a></li>
14                              <li><a href="item2/"> Item 2</a></li>
15                              <li><a href="item3/"> Item 3</a></li>
16                              <li><a href="item4/"> Item 4</a></li>
17                      </ul>
18              </div>
19      </body>
20  </html>
```

- Como detectar:

A detecção de código em Flash é bem simples, basta procurar as ocorrências das tags *object* e verificar se estas possuem um atributo *classid* com o valor igual a "clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" ou buscar por tags *embed* como atributo *type* igual a "application/x-shockwave-flash".

- Implementação da detecção:

Figura 3.29: Código XQuery que representa a busca por Flash no documento.

```
1   let $doc := doc("refatoracao3.2.4.4-flash-de.xhtml")
2   let $object := $doc//object[@classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"]
3   let $embed := $doc//embed[@type="application/x-shockwave-flash"]
4   return <flash> {$object, $embed} </flash>
```

3.3.4.5 Refatoração 10: Substituir formulários de contato por links *mailto*

E idéia neste caso é sugerir ao desenvolvedor que modifique ou acrescente a opção de enviar e-mails através de links *mailto*. Embora os formulários para envio de e-mails(exemplo da figura 3.30) sejam muito eficientes para alguns desenvolvedores, principalmente porque se pode especificar exatamente quais informações quer se obter de quem entra em contato com seu sistema, alguns desenvolvedores tem como preferência, por motivos de segurança, a utilização link *mailto*(apresentado na Figura 3.31) para envio de e-mail.

Figura 3.30: Exemplo de formulário de e-mail.

```
1   <form action="http://www.paginadeexmplo.com.br/email" method="post">
2      <input type="hidden" value="meu@email.com.br" />
3        <p>
4           <label>Assunto:
5                 <input type="text" name="asunto" />
6           </label>
7        </p>
8        <p>
9           <label>Mensagem:
10                <input type="text" name="mensagem" />
11          </label>
12       </p>
13       <p>
14          <input type="submit" value="Enviar" />
15       </p>
16  </form>
```

Figura 3.31: Exemplo da utilização de link *mailto*.

```
1   <a href="mailto:meu@email.com.br?Subject=assunto%20do%20email">
2       exemplo de envio de e-mail por mailto
3   </a>
```

- Como detectar:

Neste caso a detecção é feita conferindo todos os formulários e verificando se existem ocorrências da palavra “email” nos valores dos atributos como *name* ou *value* ou se existe o caracter ‘@’ no atributo *value* de alguma tag *input.*

- Implementação da detecção:

Figura 3.32: Consulta XQuery que verifica se possui formulários de envio de e-mail.

```
let $doc := doc("refatoracao3.2.4.5-mailto-de.xml")
let $email_form := $doc//form//*[contains( @name,"email") or contains(@value, "email")]
let $email_input := $doc//input[contains(@value, "@")]
return <ret>{$email_form, $email_input}</ret>
```

Com este foram apresentados dez algoritmos de detecção de refatorações em códigos XHTML em aplicações Web. No capítulo seguinte será apresentado um estudo de caso referente a estes algoritmos, buscando oportunidades de refatoração em um site desenvolvido em XHTML.

4. ESTUDO DE CASO

Neste capítulo será abordado o estudo de caso buscando oportunidades de refatoração nos códigos fontes de um site da Web e analisar os resultados obtidos.

4.1 SITE ESCOLHIDO

A página que foi escolhida para este estudo foi o site “Alternativas Para o Lixo Tecnológico” [e-LIXO SM, 2009], desenvolvido pelos alunos da Ciência da Computação da UFSM André Lúcio Hahn, Fabiano Romero de Azevedo, Fabrício Severo de Severo e Pablo Lima Flores inicialmente como projeto final da disciplina de Computadores e Sociedade lecionada na instituição, mas mantém-se ativo até a esta data. O objetivo do site é reunir informações sobre a coleta e separação de resíduos eletrônicos na cidade de Santa Maria, Rio Grande do Sul. Ele apresenta informações sobre o que é Lixo Tecnológico, quais são os principais componentes desses resíduos, os quais podem contaminar o solo se não forem corretamente destinados, indica a destinação correta dos materiais de acordo com o tipo e mostra locais na cidade onde a população pode enviar seu e-lixo. A figura abaixo mostra a página inicial do site.

Figura 4.1: Página inicial do site *Alternativas para o Lixo Tecnológico*.

4.2 APLICAÇÃO DAS BUSCAS DE OPORTUNIDADES DE REFATORAÇÃO

Foram escolhidas três páginas deste site para a aplicação das buscas em seu código fonte: A página que representa a explicação sobre o e-lixo [e-LIXO SM - definição, 2009] (definição.html), a página que contém a galeria de fotos [e-LIXO SM - galeria, 2009] (galeria.html) e a página de contatos [e-LIXO SM - galeria, 2009] (contatos.html).

4.2.1 Aplicando os algoritmos de detecção na página *definicao.html*

Nesta página foram aplicadas as buscas por padrões de Validade (refatoração 1 e 2) e Leiaute (refatoração 3). As Figuras 4.2, 4.3 e 4.4 apresentam os resultados.

Figura 4.2: Lista de oportunidades de refatoração 1 aplicadas na página *definição.html*.

```
Linha 74: (...) tais como <b>merc&uacute;rio</b>, <b>c&aacute;dmio</b>,
<b>ber&iacute;lio</b> e <b>chumbo</b>. Em contato (...)
```

Figura 4.3: Lista de oportunidades de refatoração 2 aplicadas na página *definição.html*.

```
Linha 71: Lixo eletr&ocirc;nico (termo que n&atilde;o deve ser confundido com <i>spam</i>),
&eacute; o nome dado aos.
```

Figura 4.4: Lista de oportunidades de refatoração 3 aplicadas na página *definição.html*.

```
Linha 80: <table class="tabela" cellpadding="1" cellspacing="1">
```

Está página possui algumas palavras que foram enfatizadas utilizando tags que podem não representar exatamente o seu sentido semântico.

Para adequar-se as recomendações atuais, é interessante alterar todas as tags *b* encontradas por *strong* e a tag *i*, neste caso por *em* pelo fato do objetivo ser apenas dar uma ênfase a palavra spam.

No caso da tabela cuja codificação inicia-se na linha 80, a refatoração dessa tabela para elementos *div* e CSS é facultativa, pois não se trata da construção do leiaute propriamente dito da página. Se for a intenção de tornar a página *tableless* (página sem

nenhum elemento *table*) então se faz necessário o uso de CSS com tags *div* configuradas de modo a simularem esta tabela.

4.2.2 Aplicando os algoritmos de detecção na página galeria.html

Nesta página foram aplicadas as buscas por padrões de Leiaute (refatoração 3) e Segurança (refatorações 7, 8 e 9).

Esta página organiza a sua galeria de fotos através de tabelas, por isso acusou nas buscas da refatoração 3, mostrada na Figura 4.5. A refatoração dessas tabelas é análoga a tabela analisada anteriormente.

Figura 4.5: Lista de oportunidades de refatoração 3 aplicadas no arquivo *galeria.html*.

```
Linha 86: <table cellpadding="0" cellspacing="1" class="retrato" id="gallery">
Linha174: <table cellpadding="0" cellspacing="1" class="retrato" id="gallery1">
```

Como a página não contém formulários e elementos Flash, não foram encontrados nenhuma ponto a ser analisado em relação às refatorações 6, 8 e 9.

4.2.3 Aplicando os algoritmos de detecção na página contatos.html

Esta página apresenta formas para o usuário entrar em contato com os desenvolvedores e alguns mapas indicando a localização de alguns lugares para recolhimento de e-lixo. Nela foram aplicadas as buscas por padrões de Acessibilidade (refatorações 4 e 5) e algumas Segurança (refatorações 7 e 10). A Figura 4.6, 4.7 e 4.8 mostras as ocorrências encontradas.

Figura 4.6: Lista de oportunidades de refatoração 4 aplicadas no arquivo *contatos.html*.

Linha 80: ** Assunto:<input** type="text" name="assunto" size="60" value="Site Alternativas para o Lixo Tecnologico"/>

Linha 81: ** Mensagem:<input** type="text" size="60" name="mensagem"/>

Linha 89: ** Assunto:<input** type="text" name="asunto" size="60" value="Site Alternativas para o Lixo Tecnologico"/>

Linha 90: ** Mensagem:**<input type="text" size="60" name="mensagem"/>

Linha 102: ** Assunto:<input** type="text" name="asunto" size="60" value="Site Alternativas para o Lixo Tecnologico"/>

Linha 103: ** Mensagem:<input** type="text" size="60" name="mensagem"/>

Linha 115: ** Assunto:<input** type="text" name="asunto" size="60" value="Site Alternativas para o Lixo Tecnologico"/>

Linha 116: ** Mensagem:<input** type="text" size="60" name="mensagem"/>

Figura 4.7: Lista de oportunidades de refatoração 7 aplicadas no arquivo *contatos.html*.

Linha 78:<form **action="www.inf.ufsm.br/~pablo/e-lixo/email"** method="post">

Linha 87:<form **action="www.inf.ufsm.br/~pablo/e-lixo/email"** method="post">

Linha 100:<form **action="www.inf.ufsm.br/~pablo/e-lixo/email"** method="post">

Linha 113:<form **action="www.inf.ufsm.br/~pablo/e-lixo/email"** method="post">

Figura 4.8: Lista de oportunidades de refatoração 10 aplicadas no arquivo *contatos.html*.

Linha 79: **<input** type="hidden" **value**="andrel**@**inf.ufsm.br"/>

Linha 88: **<input** type="hidden" **value**="fazevedo**@**inf.ufsm.br"/>

Linha 101: **<input** type="hidden" **value**="fsevero@inf.ufsm.br"/>

Linha 114: **<input** type="hidden" **value**="pablo**@**inf.ufsm.br"/>

Algumas considerações sobre as sugestões de refatoração desta página: (i) Não houve nenhuma ocorrência de refatoração 5 nesta página. (ii) Em relação ao método de envio, uma ação interessante seria dar a opção de envio ou através do formulário ou por um link mailto que foi descrito na seção 3.2.4.5 ou simplesmente substituir os formulários por links, implicitamente também já estariam refatorando os formulários post pra get. (iii) Mantendo o formulário, com finalidade de aumentar a acessibilidade, deve-se colocar a tag label envolvendo os nomes dos campos do formulário.

4.3 CONSIDERAÇÕES SOBRE O ESTUDO DE CASO

Neste capítulo foram apresentadas algumas aplicações das buscas de oportunidades de refatoração no site Alternativas para o Lixo Tecnológico.

Através dessas buscas neste site pôde-se observar que se realizadas algumas alterações em seu código fonte pode deixá-lo com o código totalmente válido, com poder maior de acessibilidade e com um desempenho em relação a suas operações melhor, isto sem alterar as suas funcionalidades.

5. TRABALHOS RELACIONADOS

Nesta seção se abordará um trabalho relacionado que de alguma forma foi relevante ao estudo desta monografia.

A tese de doutorado do professor Eduardo Kessler Piveta [PIVETA, 2009], que trabalha com muitos conceitos relacionados à refatoração, em especial a busca de oportunidades para refatoração, de um contexto mais amplo. O principal objetivo da pesquisa que esta tese descreve é prover um processo detalhado para refatoração, incluindo mecanismos para (i) seleção e criação de modelos de qualidade, padrões de refatoração e funções heurísticas, (ii) a busca e priorização de oportunidades de refatoração, (iii) a avaliação dos efeitos da refatoração na qualidade de software e (iv) a análise de vantagens e desvantagens e a aplicação de padrões de refatoração.

6. CONCLUSÃO

Este trabalho visou auxiliar com a resolução de alguns problemas relacionados ao melhoramento de códigos fonte de aplicações Web para adequar-se aos novos padrões para apresentação de conteúdo à rede, contribuindo com o estudo do conceito de refatoração, especificadamente em páginas desenvolvidas em XHTML para Web, com alguns códigos de busca e detecção de oportunidades.

O trabalho apresentou alguns conceitos relacionados à busca de oportunidades de refatoração de código associado a sistemas Web com documento XHTML. Foram abordados para a realização deste trabalho (i) estudo sobre HTML e XHTML (ii) conceitos gerais sobre refatoração de software e depois redirecionado para a refatoração para páginas Web (iii) estudo sobre XPath e XQuery, linguagens de consulta em documentos XML e XHTML (iv) Foi apresentado um catálogo com alguns problemas comuns na criação de páginas Web, cada um com a sua sugestão de refatoração e respectivo algoritmo de detecção do problema. (v) Apresentação de um estudo de caso com um site da Web, realizando as buscas e analisando os resultados.

REFERÊNCIAS

[ALTOVA XMLSpy, 2001] - Portal da companhia Altova com conteúdo relacionado ao IDE XMLSpy. Disponível em: < http://www.altova.com/xml-editor/ > Acesso em: 20 de novembro de 2011.

[EPF ECLIPSE WIKI, 2011] - EPF Eclipse Wiki - Refatoração. Disponível em: < http://epf.eclipse.org/wikis/openuppt/openup_basic/guidances/concepts/refactoring,_Poc7IPDzEdqYgerqi84oCA.html > Acesso em: 20 de setembro de 2011.

[FORNARI, 2003] - FORNARI, Miguel Rodrigues. XML – Criação de Documentos XML e Utilização em Aplicações Práticas. CBCOMP 2003. 47 p.

[FOWLER, 1999] – Refactoring: Improving the Design of Existing code, MARTIN FOWLER, Kent Beck, John Brant, Willian Opduke, Don Roberts, 1999

[HAROLD, 2010] - Refatorando HTML – Como melhorar o projeto de aplicações Web existentes, (tradução do livro original de Elliotte Rusty Harold, *Refactoring HTML: Improving the Design of Existing Web Applications*, Bookman, 2010).

[HERWINJNEN, 1994] – **Herwinjnen**, Eric Van; **Pratical SGML**; Kluer Academics Norwell, MA, USA, 1994.

[JAVIER, 2008] - Introducción a XHTML, do portal Libros Web. Disponível em: <http://www.librosWeb.es/xhtml/pdf/introduccion_xhtml.pdf > Acesso em: 25 de setembro de 2011.

[KRUTCHEN, 2000] - KRUTCHEN, P. The Rational Unified Process: An Introduction. [S.l.]: Addison Wesley, 2000.

[NEWCOMB, 1991] – Newcomb, Steven R.; the "HyTime": hypermedia/time-based document structuring language, Neil A. Kipp, Victoria Newcomb, 1991.

[PIVETA, 2009] - PIVETA, Eduardo Kessler; Improving the Search for Refactoring Opportunities on Object-Oriented and Aspect-Oriented Software, 2009

[e-LIXO SM, 2009] – Alternativas para o Lixo Tecnológico – site informativo sobre e-lixo na cidade de Santa Maria-RS.Disponível em:<http://lixotecnologicosm.cjb.net/> Acesso em: 01 de dezembro de 2011.

[e-LIXO SM - DEFINIÇÃO Alternativas para o Lixo Tecnológico – página utilizada para o estudo de caso(menu E-Lixo).Disponível em:<http://lixotecnologicosm.cjb.net/> Acesso em: 01 de dezembro de 2011.

[e-LIXO SM - GALERIA, 2009] – Alternativas para o Lixo Tecnológico – página utilizada para o estudo de caso (menu Galeria).Disponível em:<http://lixotecnologicosm.cjb.net/> Acesso em: 01 de dezembro de 2011.

[e-LIXO SM - CONTATO, 2009] – Alternativas para o Lixo Tecnológico – página utilizada para o estudo de caso (menu Contato).Disponível em:<http://lixotecnologicosm.cjb.net/> Acesso em: 01 de dezembro de 2011.

[SOUZA, 2009] - **SOUZA, Fábio dos Santos. Monografia: Realizando consultas OLAP em documentos XML**, Universidade Federal da Bahia, 2009.

[W3C XQuery, 2011] - referência da W3c sobre XQuery. Disponível em: <http://www.w3.org/XQuery >. Acesso em: 24 de setembro de 2011.

[W3C XPath, 2011] – **referência da W3C sobre XPath**. Disponível em: <http://www.w3.org/xpath >. Acesso em: 20 de setembro de 2011.

[W3C HTML5, 2011] – **referência da W3C sobre o HTML 5**. Disponível em: <http://www.w3.org/TR/html5/>. Acesso em: 03 de outubro de 2011.

[W3C XHTML, 2011] – **referência da W3C sobre o XHTML**. Disponível em: <http://www.w3.org/TR/xhtml/>. Acesso em: 25 de setembro de 2011.

[W3C HTML, 2011] – **Referência da W3C Sobre HTML.** Disponível em: <http://www.w3.org/wiki/HTML >. Acesso em: 26 de setembro de 2011

[W3C XML, 2011] – **Referência da W3C Sobre XML.** Disponível em: <http://www.w3.org/XML/>. Acesso em: 26 de setembro de 2011

[WATHWG WEB FORM 2.0, 2011] – **Referência da WATHWG Sobre formulários Web 2.0**. Disponível em: <http://www.whatwg.org/specs/Web-forms/2004-06-27-call-for-comments/>. Acesso em: 10 de dezembro de 2011