Universidade Federal de Santa Catarina
Centro Tecnológico
Curso de Sistemas de Informação

Rafael Strecker Coelho de Souza

Avaliação de Ferramentas CMS Ruby on Rails

Florianópolis
2010

Rafael Strecker Coelho de Souza

Avaliação de Ferramentas CMS Ruby on Rails

Monografia apresentada ao Curso de Sistemas de Informação da UFSC, como requisito para a obtenção parcial do grau de BACHAREL em Sistemas de Informação.

Orientador: Lúcia Helena Martins Pacheco

Doutora em Engenharia

Florianópolis

2010

Coelho de Souza, Rafael

Avaliação de Ferramentas CMS Ruby on Rails / Rafael Coelho de Souza - 2010

xx.p

1.CMS 2. Usabilidade.. I.Título.

CDU 536.21

Rafael Strecker Coelho de Souza

Avaliação de Ferramentas CMS Ruby on Rails

Monografia apresentada ao Curso de Sistemas de Informação da UFSC, como requisito para a obtenção parcial do grau de BACHAREL em Sistemas de Informação.

Aprovado em 16 de novembro de 2010

BANCA EXAMINADORA

Lúcia Helena Martins Pacheco

Doutora em Engenharia

Christiane Gresse von Wangenheim

Doutora em Ciências da Computação

José Eduardo De Lucca

Mestre em Ciências da Computação

Aos meus pais e minha namorada.

Agradecimentos

Agradeço ao meu amigo, colega de curso, parceiro de trabalhos e orientador Eduardo Bellani, pelo apoio e sua vontade em me ajudar a concluir o trabalho.

A professora Lúcia Helena Martins Pacheco pela orientação, amizade, por toda a ajuda nesta caminhada e pela paciência, sem a qual este trabalho não se realizaria.

Ao meu amigo Wilson, que me incentivou ao concluir o curso no semestre anterior.

A minha namorada Claudia, que por tantas vezes me incentivou, ajudou e me aguentou durante a jornada deste trabalho.

E a todos meus amigos e familiares que me incentivaram neste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

A quantidade de conteúdo e informações digitais vem crescendo de forma significativa. Segundo o ISC o crescimento de domínios registrados é muito grande, chegando próximo aos 800 mil domínios.

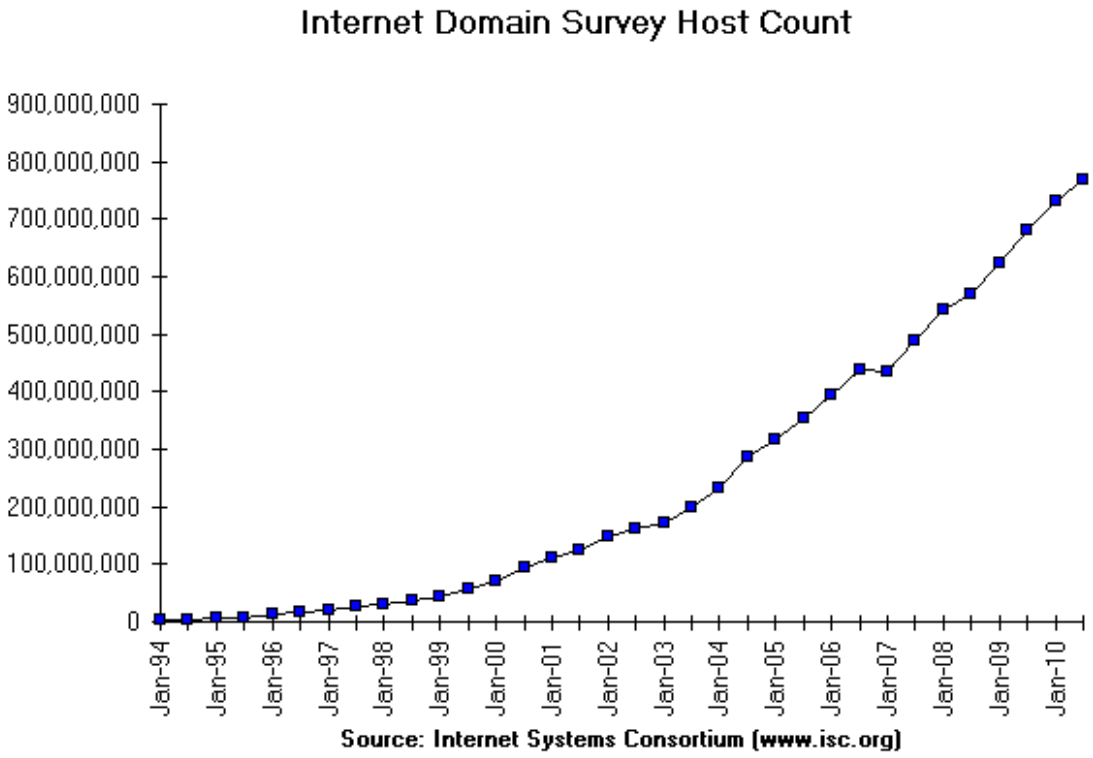

Figura 1.1: Número de domínios registrados dos últimos anos

Diante desses fatos, é intrínseco a necessidade de gerenciar e publicar esses conteúdos de forma rápida, tanto por indivíduos como organizações. A partir desta necessidade, surgem os CMS (Content Management Systems) que proporcionam versatilidade, quando o objetivo é produzir de forma rápida e organizada, soluções integradas para a exibição de conteúdo na internet.

Na escolha de um CMS para desenvolvimento deve-se pesar também não só a ferramentas mas também a plataforma a qual é desenvolvida, neste trabalho o foco será nas ferramentas da plataforma Ruby on Rails, que está se tornando muito popular nos últimos anos.

A usabilidade por sua vez, com o passar dos anos, tem recebido maior atenção, pois foi reconhecida como uma propriedade fundamental no sucesso das aplicações web.

Definir os métodos para a usabilidade é, portanto, uma das metas atuais de pesquisas. Além disso, as empresas tem reconhecido a importância dos métodos de usabilidade durante o processo de desenvolvimento, para verificar a usabilidade das aplicações, antes da sua implantação.

Fazendo uma analogia, percebe-se que gerenciamento de conteúdos e usabilidade estão diretamente relacionados, pois a usabilidade é um fator fundamental e intrínseco para o sucesso e qualidade de um projeto e satisfação do usuário, que trará benefícios significativos e tangíveis para o sistema de conteúdo.

Os aspectos de avaliação utilizados neste projeto foram a usabilidade e a comunidade open source, tendo em vista que as ferramentas avaliadas são de código aberto.

Este projeto vai avaliar as ferramentas Radiant, Refinery, BrowserCMS e Zena de acordo com os aspectos escolhidos.

1.1 Justificativa

Com a popularização do modelo open source de desenvolvimento, muitas ferramentas CMS já foram desenvolvidas e distribuídas sob licensas livres na internet. Diante deste cenário, um profissional de Sistemas de Informação deve avaliar entre as opções disponíveis e dependendo do contexto definir a utilização de uma ou outra ferramenta.

Este trabalho procura auxiliar esta avaliação, abordando a usabilidade e a comunidade ao redor das ferramentas CMS escritas em Ruby on Rails, como parâmetros para a avaliação.

1.2 Objetivos Gerais

Aprofundar o conhecimento em CMS, usabilidade e nas ferramentas avaliadas em Ruby on Rails, traçando um comparativo entre elas sob a ótica da usabilidade e da comunidade em torno das ferramentas.

1.3 Objetivos Específicos

1. Analisar as qualidades e limitações das ferramentas CMS avaliadas.

2. Utilizar técnicas de usabilidade como a inspeção por checklist para avaliar as ferramentas

3. Analisar a comunidade open source em torno da ferramenta, avaliando a atitividade recente e os canais de comunicação que são oferecidos para os desenvolvedores.

2 Estrutura do Trabalho

Nos próximos capítulos, são apresentados um aprofundamento sobre Ruby, Ruby on Rails,CMS e Usabilidade. Após, são descritas as ferramentas e a metodologia empregada na avaliação. Em seguida é feita a avaliação entre as ferramentas e uma conclusão sobre o trabalho, com uma análise crítica e sugestões de trabalhos futuros.

3 Ruby

Ruby é uma linguagem de programação de código aberto e orientada a objetos criado por Yukihiro 'Matz' Matsumoto. A primeira versão foi lançada no Japão em 1995. Ruby tem ganhado aceitação ao redor do mundo como uma linguagem fácil de aprender, poderosa, e expressiva, especialmente desde o advento do Ruby on Rails, um framework para aplicações voltadas para Web escrito em Ruby[1]. O núcleo de Ruby é escrito na linguagem de programação C e roda na maioria das plataformas. É uma linguagem interpretada e não compilada. Fitzgerald [2007b]

[2]

Uma descrição pouco mais detalhada sobre Ruby:

Ruby é uma linguagem de programação dinâmica com uma complexa, porém expressiva gramática e um núcleo de biblioteca de classes com uma rica e poderosa API. Ruby tem inspirações em Lisp,SmallTalk e Perl, mas usa uma gramática que facilita o entendimento de programadores Java e C. Ruby é uma linguagem orientada à objetos pura, mas também, suporta estilos de programação funcional e procedural. Ela inclui uma poderosa capacidade de metaprogramação e pode ser usada para criar linguagens de domínio específico conhecidas pelo acrônimo DSLs. **?**

[3]

Ruby é escrita em C e atualmente está disponível para as mais diversas plataformas UNIX, Mac OS X, Windows 95/98/Me/NT/2000/XP/Vista/Seven, DOS, BeOS, OS/2, .NET, Solaris (www.ruby lang.org [2009]).

O criador de Ruby, Yukihiro 'Matz' Matsumoto, teve a idéia de criar uma linguagem de programação em 1993, durante uma conversa um um colega sobre linguagens

[1] http://www.rubyonrails.org
[2] Tradução livre do autor
[3] Tradução livre do autor

de script. Através da conversa, o seu interesse cresceu muito sobre as linguagens de script, onde ele ficou impressionado pelo poder e pelas possibilidades. Como fora um fã de longa data de programação orientada à objetos, lhe pareceu , a orientação à objetos, muito adequada para linguagens de script. Então passou olhar pela rede e encontrou Perl 5, que ainda não havia sido lançada, contendo algumas características de orientação à objetos, mas ainda não era o que ele queria. Então decidiu abandonar Perl como uma linguagem de script orientada à objetos. Então ele procurou Python, era uma linguagem orientada à objetos interpretada. Mas ele não teve a sensação que era uma linguagem de script, era uma linguagem híbrida, procedural e orientada à objetos. Matz queria uma linguagem de script que seria mais poderosa que Perl e mais orientada à objetos que Python. Então ele decidiu criar a própria linguagem. O nome Ruby veio de uma brincadeira com um amigo durante o projeto de desenvolvimento da linguagem, ele queria um nome de uma pedra preciosa, a là Perl, e então o amigo sugeriu Ruby, que depois se tornou o nome oficial da linguagem. (Stewart [2001]).

A filosofia de Ruby segundo o Yukihiro 'Matz' Matsumoto: *Ruby foi projetado para fazer programadores mais felizes* Flanagan and Matsumoto [2008]página 1[4]

Abaixo um exemplo de uma classe Ruby

No exemplo que segue podemos visualizar como Ruby define uma classe e cria um objeto. A definição de classe se dá pelo comando *class*. No exemplo o nome da classe é *Pessoa*. Na linha 2 há uma declaração de um atributo da classe `attr_acessor`, esta declaração faz com que o Ruby gere automaticamente métodos de acessores e modificadores. O método de criação do objeto chamado *initialize* recebe um argumento *nome*. O exemplo ilustra uma das características de Ruby que é a possibilidade de valor padrão para um argumento. Ao criar o objeto com o método *new* pode se passar um objeto string com um nome, ou então irá ser utilizado o padrão.

Na linha 9 é criado um objeto da classe pessoa. Na linha 10 é escrito na tela do console o valor default retornado pelo método nome. Na linha 11 então é alterado o valor do atributo nome para em seguida ser escrito no console a nova string passada ao atributo.

code/exemplo_ruby.rb

```
1 class Pessoa
```

[4]Tradução livre do autor

```
2     attr_accessor :nome
3
4     def initialize(nome = "Rafael Strecker Coelho de Souza")
5       @nome = nome
6     end
7   end
8
9   pessoa = Pessoa.new
10  p pessoa.nome
11  pessoa.nome = 'Rafael'
12  p pessoa.nome
```

Em Ruby, como é uma linguagem orientada à objeto pura, tudo é um objeto. Desde números aos valores booleanos true e false ao nil (versão de Ruby para null, indicação de falta de um valor).Flanagan and Matsumoto [2008]

O exemplo abaixo demonstra que valores numéricos, valores booleanos, e o nil podem invocar o método *class* que retorna a sua classe e imprimiremos o resultado na tela do console.

code/exemplo_objetos.rb

```
1  p 1.class
2  p true.class
3  p nil.class
```

O resultado impresso na tela pelo exemplo é:

$ ruby exemplo_*objetos.rb*

Fixnum

TrueClass

NilClass

Cada linha é a resposta do nome da classe ao qual cada valor pertence. Por exemplo o número 1 é da classe *Fixnum*, o número 1.1 pertence a classe *Float* e assim por diante.

3.1 Interpretadores

3.1.1 MRI

O interpretador mais usado e conhecido de Ruby é o MRI, ou Matz Ruby Interpreter, escrito em C e a sua versão atual é 1.9.2. Além dele há outros interpretadores como JRuby, Rubinius, Maglev, MacRuby e IronRuby.

3.1.2 JRuby

JRuby é um interpretador Ruby rodando sobre a máquina virtual Java (JVM). Ele está na versão 1.5.2 que é compatível com o MRI 1.8.7.

3.1.3 Rubinius

Rubinius que foi originalmente idealizado para ser o interpretador de Ruby feito em Ruby, mas em virtude da compatibilidade com o MRI parte do seu código é escrito em C++. Está na versão 1.0.1 e é compatível ao MRI 1.8.7 *Um grande aspecto de linguages populares como Java e C, é que a maioria das funcionalidades disponíveis para os programadores é escrito na própria linguagem. O objetivo de Ribinius é adicionar Ruby à estas linguagens. Desta forma Rubystas podem facilmente adicionar funcionalidades à linguagem, arrumar defeitos, e aprender como a linguagem funciona. Onde é possível Rubinius é escrito em Ruby. Onde (ainda) não é escrito em C++.* rub[5]

3.1.4 Maglev

Maglev é o um interpretador Ruby desenvolvido pela empresa Gemstone. A empresa é conhecida pelo seu interpretador de Smalltalk, se lançou como uma plataforma alternativa, já que inclui um mecanismo de persistência de objetos distribuídos. Está em versão Alpha ainda.*Maglev é uma implementação de Ruby rápida e estável com uma integrada persistência de objetos e cache compartilhado e distribuído.* ruby.gemstone.com [2009][6]

[5]Tradução livre do autor

[6]Tradução livre do autor

3.1.5 IronRuby

IronRuby é um interpretador Ruby feito sobre a plataforma .NET. Ele era patrocinado pela Microsoft até Julho de 2010, onde a empresa cortou o último desenvolvedor remanescente do projeto. Está na versão 1.0 e com um futuro incerto, pois ainda não apareceu nenhum outro mantenedor oficial.

4 Ruby On Rails

Ruby on Rails é um framework projetado para escrever aplicações web de forma rápida e simples. Extraído do Basecamp, ferramenta de gerenciamento de projetos da 37 Signals, o criador David Heinemeier Hanson escreveou Rails com soluções comumente praticadas para problemas do mundo real. Estas soluções e decisões são o que torna o Rails prazeroso de usar, as chatas tarefas servis, já estão feitas, deixando você apenas para concentrar no seu problema. Fitzgerald [2007a][1]

Ruby on Rails é um framework escrito em Ruby, lançado em 2004, se tornou popular em 2005 à partir de uma palestra feita no Fórum Internacional de Software Livre pelo seu criador David Heinemeier Hanson, onde mostrou como criar uma weblog usando Rails em 15 minutos.

Rails foi extraído de uma aplicação real, o Basecamp, uma ferramenta de gerenciamento de projetos da empresa 37 Signals. Seu criador teve base em outros frameworks web, utilizando idéias de cada um e combinado à expressividade da linguagem Ruby, uma boa divulgação, atingiu uma grande popularidade. O framework introduziu a linguagem Ruby para o ocidente, e trouxe bastante atenção de programadores interessados nas facilidades providas pelo framework.

Uma de suas filosofias é a convenção ao invés de configuração (CoC-Convention over Configuration). David era contra os frameworks que abusavam de configurações em arquivos XML, então ele partiu para abolir o máximo possível arquivos de configuração. Mesmo os arquivos de configuração de Rails não utilizam XML e sim um outro formato de serialização chamdo YAML (Yet Another Markup Language). Outra filosofia adotada é o DRY (Don't Repeat Yourself), onde dita que a informação deve ser localizada em um único local.

Uma característica da arquitetura é o padrão MVC, Model View Controller, que ele utiliza. Este padrão é seguido pelos frameworks web mais populares. A figura abaixo 4.1 demonstra o funcionamento do MVC.

Podemos observar na figura que:

[1] Tradução livre do autor

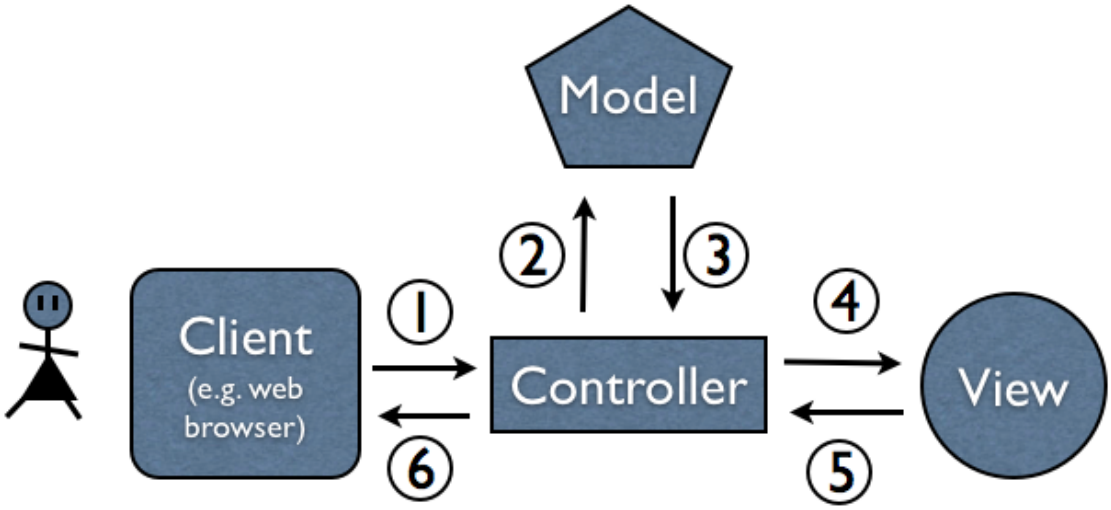

Figura 4.1: Arquitetura MVC

1. O cliente a partir do seu browser faz uma requisição para uma página.
2. Esta requisição é encaminhada ao controller, que tem a responsabilidade de determinar o destino da requisição. Ele deve pedir ao model alguns dados necessários para completar a requisição.
3. O model que é responsável pela comunicação com a base de dados, então faz uma chamada SQL para buscar os dados requisitados pelo controller e os encaminha.
4. O controller então encaminha para a view estes dados oriundos do model.
5. A view detém a responsabilidade de combinar estes dados com um modelo de html e css, gera a página html de resposta para o browser do cliente.

A estrutura do framework vem separada em diversos módulos, são eles:

- ActiveRecord - Módulo de Mapeamento Objeto Relacional, usado para facilitar o acesso e a manipulação da base de dados.
- ActionController - Módulo responsável pelo controle, usado para gerenciar todo o ciclo de vida de uma requisição.
- ActionView - Módulo responsável pela renderização dos dados, é a parte visível para o browser.
- ActionMailer - Módulo de envio de emails, age similar ao ActionView, lidando com e-mails.

- ActiveResource - Módulo de web services, auxilia a gerar e consumir web services RESTful.

Além destes módulos Rails conta uma arquitetura de plugins que fomenta a criação e extensão do framework. A partir desta arquitetura é possível extender, modificar e criar novas funcionalidades para o framework. Esta arquitetura fácil levou a criação de milhares de plugins para Rails.

Uma característica que faz do Rails uma opção popular são os seus geradores. São pedaços de códigos gerados automaticamente a partir de comandos enviados pelo console. Os geradores tornaram bem simples a geração de cadastros CRUD (Create Retrieve Update Delete), uma das atividades mais repetitivas do desenvolvimento web.

5 CMS

CMS é a sigla para Content Management System, ou sistema gerenciador de conteúdo. É a idéia de um gerenciamento da informação de organizações que produzem muito conteúdo. São baseados na web, auxiliando em vários aspectos a publicação de conteúdo, desde sua forma de apresentação ao seu controle de versão. Estes CMS servem para a publicação de conteúdo, gerenciamento de transação de e-commerces, Wikis, gerenciamento de documentos, entre outras atividades. Sistemas de Gerenciamento de Conteúdo podem de maneira simples e óbvia serem definidos por sua sigla, um sistema que gerencia conteúdos. Para uma melhor definição de CMS - Content Management Systems, o assunto será abordado na teoria de conteúdo e gestão dos mesmos.

Podemos dizer que um CMS é um framework, 'um esqueleto' de website pré-programado, com recursos básicos e de manutenção e administração já prontamente disponíveis. É um sistema que permite a criação, armazenamento e administração de conteúdo de forma dinâmica, através de uma interface de usuário via Internet. Um CMS permite que a empresa tenha total autonomia sobre o conteúdo e evolução da sua presença na internet e dispense a assistência de terceiros ou empresas especializadas para manutenções de rotina. Nem mesmo é preciso um funcionário dedicado (webmaster), pois cada membro da equipe poderá gerenciar o seu próprio conteúdo, diminuindo os custos com recursos humanos. A habilidade necessária para trabalhar com um sistema de gerenciamento de conteúdo não vai muito além dos conhecimentos necessários para um editor de texto. nav [2007]

Linguisticamente, (CMS) significa qualquer sistema que auxilia no gerenciamento de conteúdo - criação, armazenamento, indexação, arquivamento, publicação e distribuição do conteúdo. wha[1]

Em suma, o grande diferencial de um CMS é permitir que o conteúdo de um website possa ser modificado de forma rápida e segura de qualquer computador conectado à Internet. Um sistema de gerenciamento de conteúdo reduz custos e ajuda a suplantar barreiras potenciais à comunicação web reduzindo o custo da criação, contribuição e

[1] Tradução livre do autor

manutenção de conteúdo. nav [2007]

5.1 O que é Conteúdo

Há várias definições para Conteúdo, no contexto de CMS, uma visão mais simples de conteúdo vem de Kampffmeyer [2006] "O termo conteúdo representa qualquer conteúdo eletrônico, incluindo registros, dados e metadados, como também documentos e websites". A definição de Boiko [2005] "Conteúdo, portanto, é a informação que você rotula com dados para então um computador poder organizar e sistematizar a coleção, gerenciamento e publicação". Estas definições apresentam algumas diferenças, mas nota-se que sem o auxílio de uma ferramenta específica para gerenciar conteúdos, seria uma tarefa deveras árdua.

5.2 Tipos de CMS

Como CMS é um conceito amplo, existem várias classificações entre os Gerenciadores de Conteúdo. Alguns, podem ser mais específicos como o funcionamento de blogs ou wikis, outros mais generalistas como os Web Content Management Systems (WCMS).

Segundo Mehta [2009] eis as seguintes classificações

5.2.1 Portais ou CMS genéricos ou WCMS

Estes CMS são muito populares, geralmente encontrados na confecção de sites corporativos, de pequeno até grande porte no caso de Portais. Eliminam em grande parte a complexidade do site ser administrado por uma pessoa técnica, conhecida como webmaster. Com este tipo de CMS pessoas com pouco conhecimento técnico podem publicar conteúdos.

Principais Características

- Criar a gerenciar seções de conteúdos.
- Criar páginas e adicionar conteúdos de textos ou imagens.

- Editar conteúdo publicado.
- Administração por múltiplos usuários.
- Versionamento de conteúdo.
- Gerenciamento de Workflow.

Exemplos de WCMS

- BrowserCMS
- RadiantCMS
- RefineryCMS
- ZenaCMS
- Drupal
- Joomla
- Liferay

5.2.2 Blog CMS

Blogs também são considerados CMS pois são autorais, publicam conteúdo de texto, imagem e vídeos.

Principais Características

- Criar posts.
- Categorizar Posts.
- Gerenciar comentários.
- Adicionar imagens, vídeos e textos.

Exemplos de Blog CMS

- WordPress
- Mephisto
- Typo
- Blogger

5.2.3 Wiki CMS

Wiki é uma página ou coleção de páginas web projetadas para permitir o acesso, a qualquer usuário, para contribuir ou modificar o conteúdo (excluindo os usuários bloqueados), utilizando uma linguagem de marcação simplificada. Wikis são geralmente usados para criarem sites colaborativos e ampliar sitemas de comunidades. Mehta [2009]página 22[2]

Principais Características

- Facilidade de criar páginas, chamadas de wikiweb.
- Linguagem de marcação simples.
- Criação de links automatizados, mesmo que o link ainda não exista.
- Sistema completo de versionamento.
- Pode ter acesso restrito à usuários ou grupos.

Exemplos de Wiki CMS

- MediaWiki
- TWiki
- Typo
- Blogger

[2]Tradução livre do autor

5.2.4 eLearning CMSs

eLearning CMS são gerenciadores de conteúdos especializados em ensino à distância. Também conhecidos como LMS Learning Management Systems, eles têm responsabilidade pela administração, documentação, registro e relatório de cursos.

Principais Características

- Gerenciar cursos, estudantes, professores.
- Criar um curso e um programa de aprendizado.
- Criar documentos, testes, discussões e anúncios.
- Sistema de chat, forum, blogs, etc.

Exemplos de eLearning CMS

- Dokeos
- Moodle
- LAMS

Mehta [2009][3]

[3]Tradução livre do autor

6 Usabilidade

Por muitos anos, as interfaces encontradas, eram de difícil manuseio e confusas o que repelia parte de seus usuários, muitos quando se deparavam com tais ferramentes não sentiam-se confortáveis e abandonavam.

Mas, com o alto crescimento de usuários das ferramentes web, viu-se a necessidade de criar CMS mais simples que facilitassem a criação e manutenção de sites, pois o usuário precisa conseguir utilizar e desejar usar novamente, e isso só acontecerá se ele encontrar o que procura com facilidade.

> Se a interface com o usuário for muito rígida, lenta, desagradável, pessoas se sentem frustradas, abandonam e esquecem o produto. C. Ardito [2002][1]

A norma da International Organization for Standardization 9241 define usabilidade como "É a medida pela qual um produto pode ser usado por usuários específicos para alcançar objetivos específicos com efetividade, eficiência e satisfação em um contexto de uso específico"

> A usabilidade visa impactar positivamente sobre o retorno do investimento para a empresa. Ela será argumento de vendas, passará uma imagem de qualidade, evitará prejuízos para os clientes, ligados ao trabalho adicional e ao "retrabalho"de correções freqüentes, por exemplo. A empresa desenvolvedora economizará custos de manutenção e de revisões nos produtos, como mostra o texto sobre Engenharia de Usabilidade. nie [b]

Sistemas que visam uma boa usabilidade tem como dever fomentar a criação de interfaces simples, de modo a não dificultar os processos, ajudando o usuário a ter controle de todo o ambiente sem ser obstrusiva.

Projetar visando a usabilidade envolve estabelecer os requisitos de usuários para um novo sistema ou produto, desenvolver soluções de desing, protótipos do sistema e da interface com o usuário, e testar com os usuários significativos. No entanto, antes

[1] Tradução livre do autor

de qualquer atividade de projeto ou avaliação de usabilidade começar, é necessário compreender o Contexto de uso do produto, exemplificando, os objetivos da comunidade de usuários, o usuário principal, tarefas e as característias do ambiente em que o sistema irá operar. MAGUIRE [2001][2]

Interfaces com baixa qualidade de uso trazem diversos problemas, dentre os quais: confusão aos usuários, treinamento excessivo, desmotivação a exploração, indução ao erro, entre outras. Estes problemas podem ser detectados por diversos métodos de avaliação, realizados ao longo do processo de desenvolvimento. Os métodos de avaliação mais utilizados se concentram em avaliar a usabilidade de um sistema. MAGUIRE [2001][3]

A usabilidade pode ser dividida em cinco princípios básicos

- Intuitividade: O sistema deve apresentar facilidade de uso permitindo que, mesmo um usuário sem experiência, seja capaz de produzir algum trabalho satisfatoriamente.
- Eficiência: O sistema deve ser eficiente em seu desempenho apresentando um alto nível de produtividade.
- Memorização: Suas telas devem apresentar facilidade de memorização permitindo que usuários ocasionais consigam utilizá-lo mesmo depois de um longo intervalo de tempo.
- Erro: A quantidade de erros apresentados pelo sistema deve ser o mais reduzido possível, além disso, eles devem apresentar soluções simples e rápidas mesmo para usuários iniciantes. Erros graves ou sem solução não podem ocorrer.
- Satisfação: O sistema deve agradar o usuário, sejam eles iniciantes ou avançados, permitindo uma interação agradável.

nie [b]

A partir destes princípios, é possivel aprofundá-los e especializar em alguns critérios que podem ser avaliados entre diferentes métodos de avaliação.

Existem algumas maneiras de se avaliar a usabilidade e ergonomia de um sistema, explorando os critérios a partir dos princípios acima descritos.

[2]Tradução livre do autor

[3]Tradução livre do autor

6.1 Métodos de Avaliação

Se a interface não tem uma boa qualidade, muitos problemas podem ser desencadeados. As mesmas podem ser confusas ao usuário, o que gerará muito treinamento, também a desmotivação em explorar as ferramentas o que induzirá ao erro, entre outros aspectos. Esses problemas podem ser detectados por diversos métodos de avaliação, que são realizados ao longo do processo de desenvolimento, esses concentram-se em avaliar a usabilidade do sistema.

Avaliação de Usabilidade é um nome genérico para um conjunto de métodos que são baseados em avaliadores inspecionando a interface. Tipicamente, inspeção de usabilidade é direcionado a procurar problemas de usabilidade em uma interface. Vários métodos de inspeção focam nas especificações de interface com o usuário, que podem não estar necessariamente implementada, isso significa que a inspeção pode ser feita em estágios primários do ciclo de vida da engenharia de usabilidade. nie [a]

Há 3 principais técnicas de avaliação de ergonomia. São elas:

- Técnicas Prospectivas
- Técnicas Preditivas ou Diagnósticas
- Técnicas Objetivas ou Empíricas

6.1.1 Técnicas Prospectivas

Este tipo de técnica está baseada na aplicação de questionários/entrevistas com o usuário para avaliar sua satisfação ou insatisfação em relação ao sistema e sua operação. Ela mostra-se bastante pertinente na medida em que é o usuário a pessoa que melhor conhece o software, seus defeitos e qualidades em relação aos objetivos em suas tarefas. Nada mais natural em buscar suas opiniões para orientar revisões de projeto.

Muitas empresas de software elaboram e aplicam regularmente este tipo de questionário, como parte de sua estratégia de qualidade, porém constatou-se que os questionários de satisfação têm uma taxa de devolução reduzida (máximo 30% retornam), o que indica a necessidade de elaboração de um questionário mais dinâmico, com questões

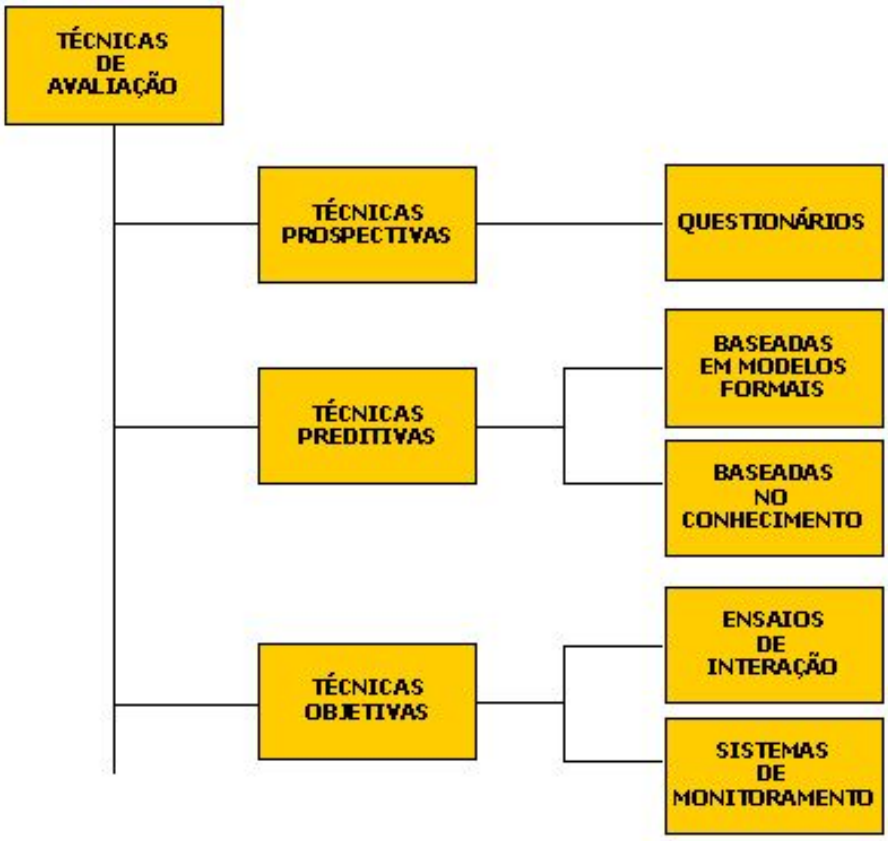

Figura 6.1: Diagrama das técnicas de usabilidade

mais sucintas e diretas e que tenham espaço para opiniões e sugestões, pois questionários longos, tornam-se cansativos. cyb

As técnicas prospectivas mais comuns são: questionários de opinão dos usuários, registros de uso do sistema e coleta de opiniões de especialistas.

6.1.2 Técnicas Preditivas ou Diagnósticas

Técnicas preditivas são baseadas em avaliações de especialistas, e na competência destes avaliadores.

As técnicas diagnósticas dispensam a participação direta de usuários nas avaliações, que se baseiam em verificações e inspeções de versões intermediárias ou acabadas de software interativo, feitas pelos projetistas ou por especialistas em usabilidade. cyb

As avaliações podem ser divididas em:

6.1.3 Avaliações Analíticas

Essa técnica é empregada nas primeiras etapas da concepção de interfaces humano-computador, quando ela não passa de uma descrição da organização das tarefas interativas. Mesmo nesse nível, já é possível verificar questões como a consistência, a carga de trabalho e o controle do usuário sobre o diálogo proposto e a realizar. A especificação da futura tarefa interativa, pode ser realizada nos termos de um formalismo apropriado como MAD, GOMS (Goals, Operators, Methods and Selections rules) e CGL (Command Grammar Language).

Em particular, GOMS propõe uma tabela associando tempos médios de realização aos métodos primitivos, que correspondem as primitivas ações físicas ou cognitivas. Com base na descrição da tarefa realizada segundo o formalismo é possível calcular os tempos prováveis para a realização das tarefas previstas. cyb

As avaliações analíticas se dividem em métodos formais e aproximados.O método formal exige inspeção cuidadosa de seqüências de ação que um usuário realiza para concluir uma tarefa. Isto também é chamado "keystrokelevel analysis". Isto envolve dividir a tarefa em ações individuais, como movimentar o mouse para o menu ou o digitar no teclado e calcular o tempo que leva estas ações.

O método aproximado é menos detalhado e provém resultados menos precisos, porém podem ser efetuados de maneira muito mais rápida. Envolve um processo similar de sequência de ações que um usuário realiza nos quesitos físico, cognitivo e perceptivo.

As vantagens de avaliações analíticas estão em uma predição precisa do quanto tempo leva a execução de uma tarefa e uma análise profunda do comportamento do usuário durante a mesma.

As desvantanges são o tempo e custo disto e ainda requerem avaliadores de alta capacidade. Holzinger [2005]

Avaliações Heurísticas

Uma avaliação heurística é um método de análise de interfaces e qualidades ergonômicas das interfaces humano-computador, com base no conjunto de critérios de usabilidade. Os princípios são chamados de heurísticas pois são desenvolvidos a partir de uma série de experiências prévias, sintetizando pontos recorrentes. Essa avaliação é realizada por

especialistas em ergonomia, baseados em sua experiência e competência no assunto. Eles examinam o sistema interativo e diagnosticam os problemas ou as barreiras que os usuários provavelmente encontrarão durante a interação. cyb

Avaliação heurística é o mais informal de métodos de inspecção. Ela é formada por um pequeno conjunto de especialistas, para analisar a aplicação através de uma lista de princípios de usabilidade reconhecidos - a heurística. Ela é um método muito eficiente de engenharia de usabilidade,com uma relação custo-benefício alta.

Jakob Nielsen sugere um conjunto de 10 regras heurísticas para guiar uma avaliação, são elas:

- Diálogos simples e naturais: O sistema deve apresentar facilidade de uso permitindo que, mesmo um usuário sem experiência, seja capaz de produzir algum trabalho satisfatoriamente.
- Saídas claramentes definidas: O usuário controla o sistema e pode a qualquer momento abortar uma função ou tarefa indesejada e retonar ao estado anterior.
- Atalhos: Para usuários experientes executarem as tarefas mais rapidamente.
- Consistência: Um mesmo comando deve ter o mesmo efeito sempre.
- Feedback: O sistema deve informar continuamente ao usuário o que ele está fazendo, através de uma resposta em um tempo curto.
- Memorização: Suas telas devem apresentar facilidade de memorização permitindo que usuários ocasionais consigam utilizá-lo mesmo depois de um longo intervalo de tempo.
- Prevenção de erros: A quantidade de erros apresentados pelo sistema deve ser o mais reduzido possível.
- Boas mensagens de erro: Linguagem clara, devem ajudar o usuário a entender o problema, não devem intimidar o usuário.
- Satisfação: O sistema deve agradar os usuários, sejam eles iniciantes ou avançados, permitindo uma interação agradável.
- Ajuda e documentação: O ideal é que o sistema seja tão intuitivo que não precise de ajuda. Se for necessária, deve ser de fácil acesso online.

Inspeções por Checklist

As inspeções de usabilidade por checklists, são vistorias baseadas em listas de verificação, através das quais profissionais, não necessariamente especialistas em ergonomia, como por exemplo, programadores e analistas, diagnosticam rapidamente problemas gerais e repetitivos das interfaces. Neste tipo de técnica, ao contrário das avaliações heurísticas, são as qualidades da ferramenta (checklists) e não dos avaliadores, que determinam as possibilidades para a avaliação. Checklists bem elaborados devem produzir resultados mais uniformes e abrangentes, em termos de identificação de problemas de usabilidade, pois os inspetores são conduzidos no exame da interface através de uma mesma lista de questões a responder sobre a usabilidade do projeto. cyb

6.1.4 Técnicas Objetivas ou Empíricas

As técnicas objetivas, se baseiam na participação direta de usuários através de duas principais formas:

- Ensaios de interação
- Sistemas de Monitoramento de interação

7 Ferramentas Avaliadas

As ferramentas avaliadas pertencem ao grupo dos Web Content Management Systems, que serão avaliadas dentro deste contexto elas.

7.1 RadiantCMS

> Radiant é um sistema de gerenciamento de conteúdo aberto, simples e projetado para pequenas equipes rad[1]

Radiant é um dos CMS mais antigos disponíveis para Ruby, lançado em 2006. De instalação e uso simples tem uma grande quantidade de plugins disponíveis e uma grande comunidade mantenedora da aplicação.

7.1.1 Características

- Interface intuitiva
- Bom sistema de extensões, com um número muito grande de extensões disponíveis.
- Sistema de usuários e permissões simples.
- Snippets
- Linguagem específica de domínio Radius.

Radiant tem um sistema de usuários e permissões simplificado, bom para pequenas organizações que não necessitam de grande complexidade no acesso. Possui algumas extensões de código aberto que adicionam um maior leque de permissões e funcionalidades ao sistema.

Os Snippets são pedaços reutilizáveis de código, que podem ser inseridos em qualquer página ou template (layout), são úteis e dentro da filosofia de Rails DRY, Don't Repeat Yourself.

[1] Tradução livre do autor

Figura 7.1: Interface de administração do Radiant

Radius é uma linguagem específica de domínio escrita em Ruby, baseada em tags, semelhante a XML e HTML. É utilizada dentro do Radiant para prover algumas funcionalidades como os snippets e outras marcações que encapsulam parte da lógica da aplicação. É possível a partir dela gerar qualquer forma de texto puro ou html. Radiant utiliza várias tags Radius para auxiliar no desenvolvimento de uma aplicação como:

- Tags que geram conteúdos: `<r:author />`,`<r:breadcrumb />`, `etc.`
- Tags de controle de fluxo `<r:if_parent />`,`<r:unless_parent />`,`etc.`
- Tags que modificam o contexto da página `<r:page>`, `<r:children>` ,`etc.`
- Tags que trabalham com coleções `<r:children:each >`

7.2 RefineryCMS

RefineryCMS é um sistema gerenciador de conteúdos desenvolvido pela Resolve Digital. Ele é um sistema que começou em 2005 e que em meados de 2009 abriu o seu código para a comunidade. Projetado para pequenos projetos onde pessoas possam manter o conteúdo sem muita complexidade. Possui uma interface amigável, simples de se trabalhar, com um editor de conteúdo WYISWYM (What You See Is What You Mean) integrado. Sua instalação é simples e a documentação é pequena porém bem organizada. Ele recentemente foi portado ja para Ruby on Rails versão 3 e utiliza engines, que são mini aplicações Rails dentro de um projeto Rails, como forma de extensão, facilitando para a comunidade escrever extensões para a ferramenta.

O sistema de modelos (templates) do Refinery é feito através de temas, gerados a partir de seu modelo de extensões. Seu ponto forte é que não é necessário aprender uma linguagem específica de domínio como o Radius para efetuar as customizações de layout. Uma desvantagem é que o usuário final pouco pode customizar um layout dentro da interface de administração.

7.2.1 Características

- Interface intuitiva, fácil para usuários não técnicos.
- Sistema de extensões baseado em engines, fácil de ser usado, porém não há muitas engines disponíveis.
- Instalação simples
- Documentação bem organizada.
- O foco do usuário final é somente gerenciar conteúdo.

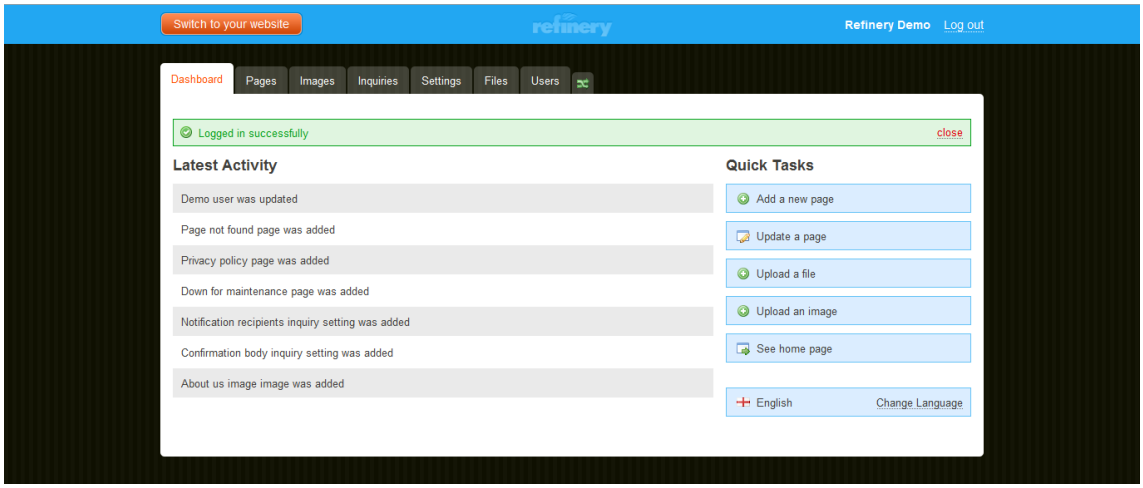

Figura 7.2: Interface de administração do Refinery

7.3 BrowserCMS

BrowserCMS é um sistema gerenciador de conteúdos desenvolvido pela empresa Browser-Media, econtra-se na versão 3.1.2. Foi por vários anos um CMS comercialmente licensiado, escrito na linguagem Java. Motivados pela "onda"de Ruby on Rails, e pela falta de um

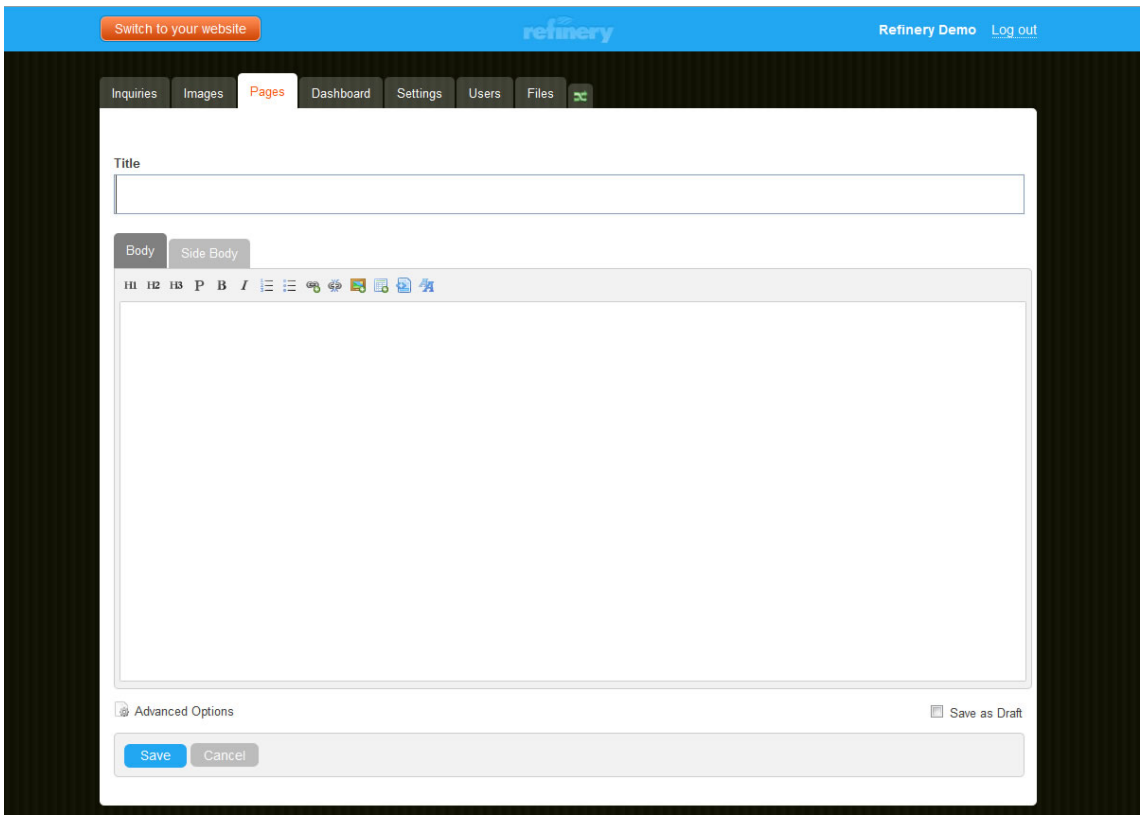

Figura 7.3: Interface de criação / edição de página do Refinery

CMS em Ruby on Rails que suprisse as necessidades de seus clientes, reescreveram o seu CMS em 2009 e deixaram-o com uma licensa de código aberto.

7.3.1 Características

- Sistema voltado ao mundo corporativo para grandes e médias organizações.
- Sistema de workflow completo de publicação.
- Sistema amplo de permissões.
- Versionamento de conteúdo.
- Poucas extensões
- Documentação pequena.

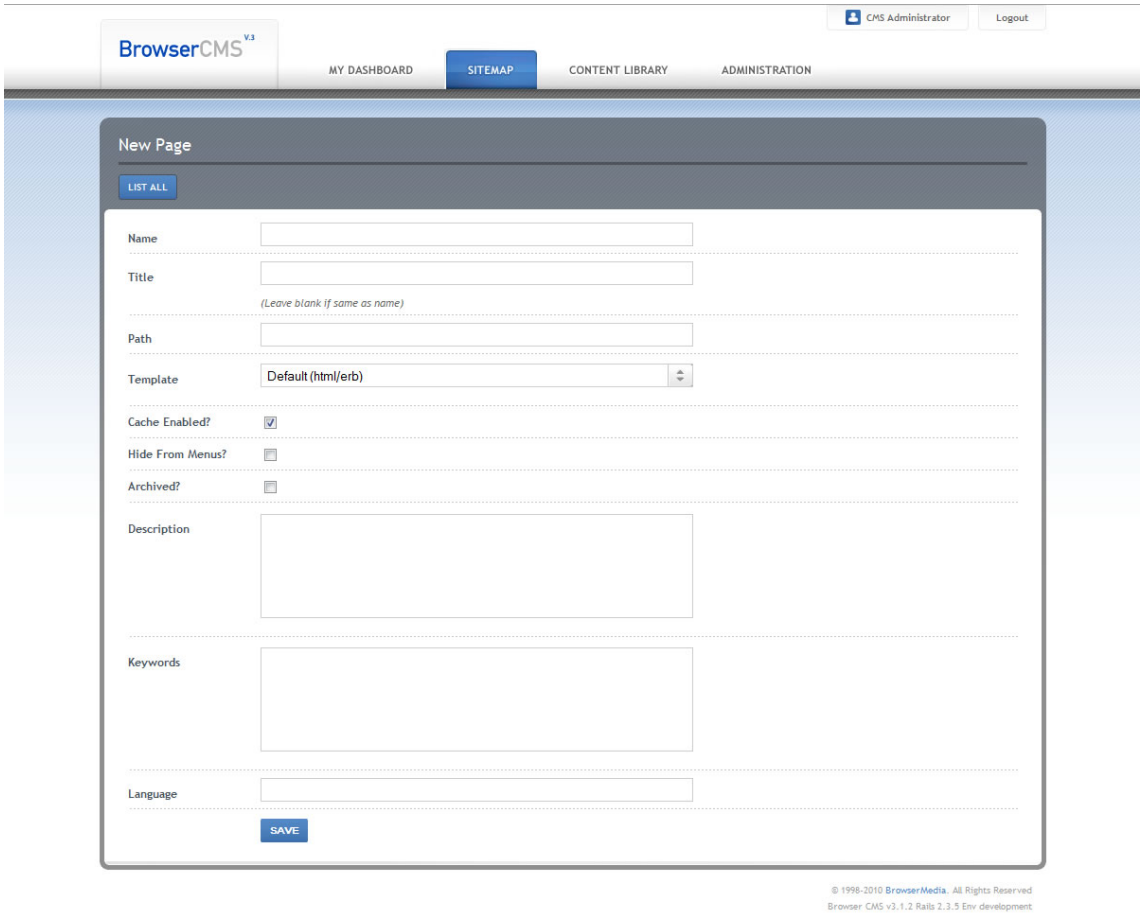

Figura 7.4: Interface de criação de página do BrowserCMS

7.4 Zena

Zena é um CMS baseado em Ruby on Rails, escrito a partir de 2007 por Gaspard Bucher que inicialmente o desenvolveu para o seu uso pessoal. Após um certo tempo ele decidiu incrementar a flexibilidade do CMS, adicionando uma linguagem de templates, Zafu, que poderia ser usada também por outras pessoas.

A linguagem específica de domínio Zafu, tem objetivo semelhante ao Radius, de abstrair e facilitar a inclusão de códigos ruby dentro das páginas criadas.

7.4.1 Características

- Sistema voltado ao mundo corporativo para grandes e médias organizações.
- Sistema de workflow completo de publicação.

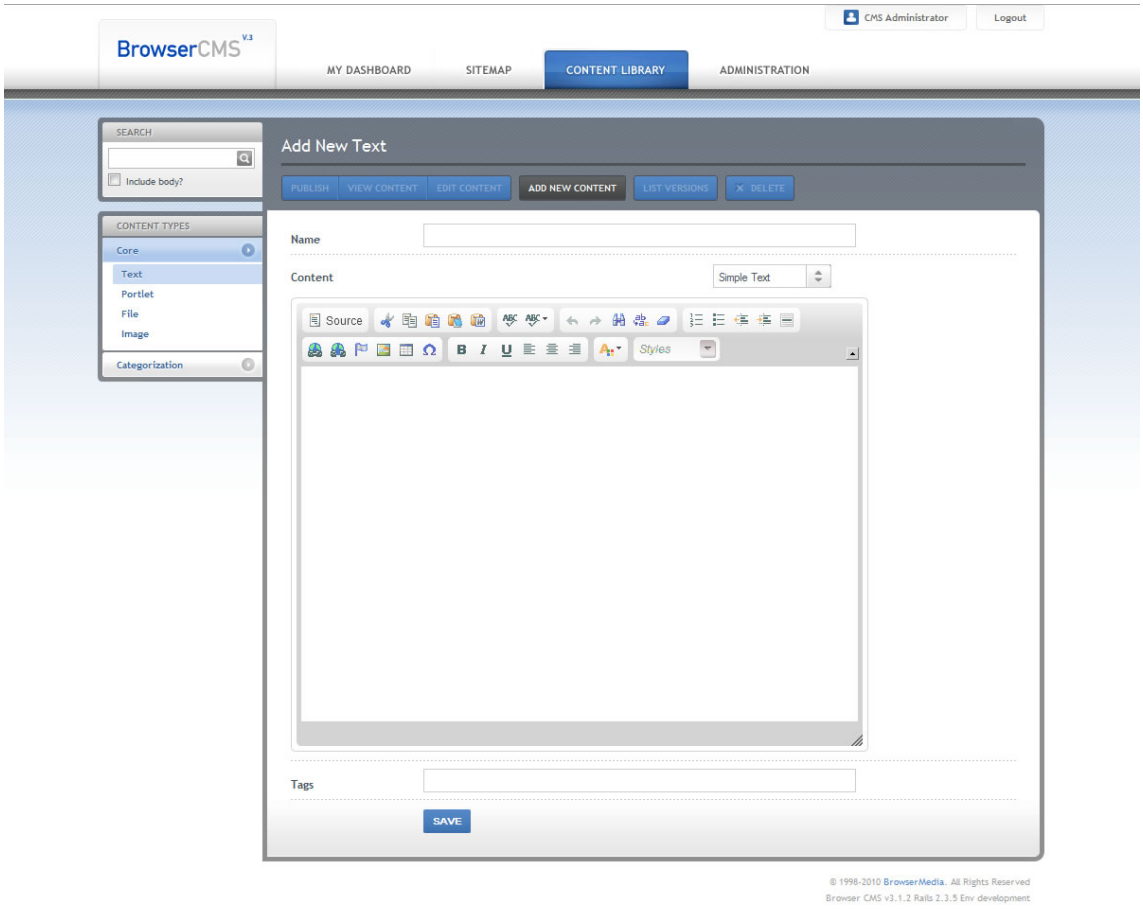

Figura 7.5: Interface de criação de conteúdo do BrowserCMS

- Sistema amplo de permissões.
- Versionamento de conteúdo.
- Poucas extensões
- Documentação pequena.

8 Metodologia

Este capítulo se dedica a informar quais as serão os métodos de avaliação entre as ferramentas CMS.

8.1 Inspeção por Checklist

Este método foi selecionado dentre os apresentados anteriormente, pois pode ser aplicado por uma pessoa sem muita experiência com usabilidade, como o próprio autor deste trabalho, nas ferramentas selecionadas.

A inspeção adotada é uma checklist mais voltada a websites, adaptadas as necessidades das ferramentas avaliadas. A inspeção foi retirada do site `http://www.userfocus.co.uk/resources/navchecklist.html`.Os critérios avaliados a seguir, não foram traduzidos pois poderiam perder um pouco do significado.

- home page usability guidelines

    1. The items on the home page are clearly focused on users' key tasks ("featuritis"has been avoided).
    2. If the site is large, the home page contains a search input box.
    3. Useful content is presented on the home page or within one click of the home page.
    4. Links on the home page begin with the most important keyword (e.g. "Sun holidays"not "Holidays in the sun").
    5. There is a short list of items recently featured on the homepage, supplemented with a link to archival content.
    6. Navigation areas on the home page are not over-formatted and users will not mistake them for adverts.
    7. The value proposition is clearly stated on the home page (e.g. with a tagline or welcome blurb).

8. Navigation choices are ordered in the most logical or task-oriented manner (with the less important corporate information at the bottom).
9. The title of the home page will provide good visibility in search engines like Google.
10. All corporate information is grouped in one distinct area (e.g. "About Us").
11. Users will understand the value proposition.
12. By just looking at the home page, the first time user will understand where to start.
13. The home page shows all the major options.
14. The home page of the site has a memorable URL.
15. The home page is professionally designed and will create a positive first impression.
16. The design of the home page will encourage people to explore the site.
17. The home page looks like a home page; pages lower in the site will not be confused with it.

- Task Orientation

1. The site is free from irrelevant, unnecessary and distracting information.
2. Excessive use of scripts, applets, movies, audio files, graphics and images has been avoided.
3. The site avoids unnecessary registration.
4. The critical path (e.g. purchase, subscription) is clear, with no distractions on route.
5. Information is presented in a simple, natural and logical order.
6. The number of screens required per task has been minimised.
7. The site requires minimal scrolling and clicking.
8. The site correctly anticipates and prompts for the user's probable next activity.
9. Users can complete common tasks quickly.
10. The site makes the user's work easier and quicker than without the system.

11. The most important and frequently used topics, features and functions are close to the centre of the page, not in the far left or right margins.
12. The user does not need to enter the same information more than once.
13. Important, frequently needed topics and tasks are close to the 'surface' of the web site.
14. Typing (e.g. during purchase) is kept to an absolute minimum, with accelerators ("one-click") for return users.
15. The path for any given task is a reasonable length (2-5 clicks).
16. When there are multiple steps in a task, the site displays all the steps that need to be completed and provides feedback on the user's current position in the workflow.
17. The use of metaphors is easily understandable by the typical user.
18. Data formats follow appropriate cultural conventions (e.g. miles for UK).
19. Details of the software's internal workings are not exposed to the user.
20. The site caters for users with little prior experience of the web.
21. A typical first-time visitor can do the most common tasks without assistance.
22. When they return to the site, users will remember how to carry out the key tasks.
23. Action buttons (such as "Submit") are always invoked by the user, not automatically invoked by the system when the last field is completed.
24. Command and action items are presented as buttons (not, for example, as hypertext links).
25. If the user is half-way through a transaction and quits, the user can later return to the site and continue from where he left off.
26. When a page presents a lot of information, the user can sort and filter the information.
27. If there is an image on a button or icon, it is relevant to the task.
28. The site prompts the user before automatically logging off the user, and the time out is appropriate.

29. The site is robust and all the key features work (i.e. there are no javascript exceptions, CGI errors or broken links).
30. The site supports novice and expert users by providing different levels of explanation (e.g. in help and error messages).
31. The site allows the user to customise operational time parameters (e.g. time until automatic logout).

- Navigation and IA

1. There is a convenient and obvious way to move between related pages and sections and it is easy to return to the home page.
2. The information that users are most likely to need is easy to navigate to from most pages.
3. Navigation choices are ordered in the most logical or task-oriented manner.
4. The navigation system is broad and shallow (many items on a menu) rather than deep (many menu levels).
5. The site structure is simple, with a clear conceptual model and no unnecessary levels.
6. The major sections of the site are available from every page (persistent navigation) and there are no dead ends.
7. Navigation tabs are located at the top of the page, and look like clickable versions of real-world tabs.
8. There is a site map that provides an overview of the site's content.
9. The site map is linked to from every page.
10. The site map provides a concise overview of the site, not a rehash of the main navigation or a list of every single topic.
11. Good navigational feedback is provided (e.g. showing where you are in the site).
12. Category labels accurately describe the information in the category.
13. Links and navigation labels contain the "trigger words"that users will look for to achieve their goal.

14. Terminology and conventions (such as link colours) are (approximately) consistent with general web usage.
15. Links look the same in the different sections of the site.
16. The terms used for navigation items and hypertext links are unambiguous and jargon-free.
17. There is a visible change when the mouse points at something clickable (excluding cursor changes).
18. Hypertext links that invoke actions (e.g downloads, new windows) are clearly distinguished from hypertext links that load another page.
19. The site allows the user to control the pace and sequence of the interaction.
20. There are clearly marked exits on every page allowing the user to bale out of the current task without having to go through an extended dialog.
21. The site does not disable the browser's "Back"button and the "Back"button appears on the browser toolbar on every page.
22. Clicking the back button always takes the user back to the page the user came from.
23. If the site spawns new windows, these will not confuse the user (e.g. they are dialog-box sized and can be easily closed).
24. Menu instructions, prompts and messages appear on the same place on each screen

- Forms and data entry

1. Fields in data entry screens contain default values when appropriate and show the structure of the data and the field length.
2. Field labels on forms clearly explain what entries are desired.
3. Text boxes on forms are the right length for the expected answer.
4. There is a clear distinction between "required"and "optional"fields on forms.
5. Questions on forms are grouped logically, and each group has a heading.
6. Fields on forms contain hints, examples or model answers to demonstrate the expected input.

7. When field labels on forms take the form of questions, the questions are stated in clear, simple language.
8. Pull-down menus, radio buttons and check boxes are used in preference to text entry fields on forms (i.e. text entry fields are not overused).
9. With data entry screens, the cursor is placed where the input is needed.
10. Users can complete simple tasks by entering just essential information (with the system supplying the non-essential information by default).
11. Forms allow users to stay with a single interaction method for as long as possible (i.e. users do not need to make numerous shifts from keyboard to mouse to keyboard)..
12. Text entry fields indicate the amount and the format of data that needs to be entered.
13. Forms are validated before the form is submitted .
14. With data entry screens, the site carries out field-level checking and form-level checking at the appropriate time.
15. The site makes it easy to correct errors (e.g. when a form is incomplete, positioning the cursor at the location where correction is required).
16. There is consistency between data entry and data display.
17. Labels are close to the data entry fields (e.g. labels are right justified)

- Page layout and visual design

1. The screen density is appropriate for the target users and their tasks.
2. The layout helps focus attention on what to do next.
3. On all pages, the most important information (such as frequently used topics, features and functions) is presented on the first screenful of information ("above the fold").
4. The site can be used without scrolling horizontally.
5. Things that are clickable (like buttons) are obviously pressable.
6. Items that aren't clickable do not have characteristics that suggest that they are.

7. The functionality of buttons and controls is obvious from their labels or from their design.
8. Clickable images include redundant text labels (i.e. there is no 'mystery meat' navigation).
9. Hypertext links are easy to identify (e.g. underlined) without needing to 'minesweep'.
10. Fonts are used consistently.
11. The relationship between controls and their actions is obvious.
12. Icons and graphics are standard and/or intuitive (concrete and familiar).
13. There is a clear visual "starting point"to every page.
14. Each page on the site shares a consistent layout.
15. Buttons and links show that they have been clicked.
16. GUI components (like radio buttons and check boxes) are used appropriately .
17. Fonts are readable.
18. The site avoids italicised text and uses underlining only for hypertext links.
19. There is a good balance between information density and use of white space.
20. The site is pleasant to look at.
21. Pages are free of "scroll stoppers"(headings or page elements that create the illusion that users have reached the top or bottom of a page when they have not).
22. The site avoids extensive use of upper case text.
23. The site has a consistent, clearly recognisable look and feel that will engage users.
24. Saturated blue is avoided for fine detail (e.g. text, thin lines and symbols).
25. Colour is used to structure and group items on the page.
26. Emboldening is used to emphasise important topic categories .
27. Pages have been designed to an underlying grid, with items and widgets aligned both horizontally and vertically.
28. Meaningful labels, effective background colours and appropriate use of borders and white space help users identify a set of items as a discrete functional block.

29. The colours work well together and complicated backgrounds are avoided.
30. Individual pages are free of clutter and irrelevant information.
31. Standard elements (such as page titles, site navigation, page navigation, privacy policy etc.) are easy to locate.
32. The organisation's logo is placed in the same location on every page, and clicking the logo returns the user to the most logical page (e.g. the home page).
33. Attention-attracting features (such as animation, bold colours and size differentials) are used sparingly and only where relevant.
34. Icons are visually and conceptually distinct yet still harmonious (clearly part of the same family).
35. Related information and functions are clustered together, and each group can be scanned in a single fixation (5-deg, about 4.4cm diameter circle on screen).

- Search Usability

1. The default search is intuitive to configure (no Boolean operators).
2. The search results page shows the user what was searched for and it is easy to edit and resubmit the search.
3. Search results are clear, useful and ranked by relevance.
4. The search results page makes it clear how many results were retrieved, and the number of results per page can be configured by the user.
5. If no results are returned, the system offers ideas or options for improving the query based on identifiable problems with the user's input.
6. The search engine handles empty queries gracefully.
7. The most common queries (as reflected in the site log) produce useful results.
8. The search engine includes templates, examples or hints on how to use it effectively.
9. The site includes a more powerful search interface available to help users refine their searches (preferably named "revise search"or "refine search", not "advanced search").

10. The search results page does not show duplicate results (either perceived duplicates or actual duplicates).
11. The search box is long enough to handle common query lengths.
12. Searches cover the entire web site, not a portion of it.
13. If the site allows users to set up a complex search, these searches can be saved and executed on a regular basis (so users can keep up-to-date with dynamic content).
14. The search interface is located where users will expect to find it (top right of page).
15. The search box and its controls are clearly labeled (multiple search boxes can be confusing).
16. The site supports people who want to browse and people who want to search.
17. The scope of the search is made explicit on the search results page and users can restrict the scope (if relevant to the task).
18. The search results page displays useful meta-information, such as the size of the document, the date that the document was created and the file type (Word, pdf etc.).
19. The search engine provides automatic spell checking and looks for plurals and synonyms.
20. The search engine provides an option for similarity search ("more like this").

- Help, feedback and error tolerance

1. The FAQ or on-line help provides step-by-step instructions to help users carry out the most important tasks.
2. It is easy to get help in the right form and at the right time.
3. Prompts are brief and unambiguous.
4. The user does not need to consult user manuals or other external information to use the site.
5. The site uses a customised 404 page, which includes tips on how to find the missing page and links to "Home"and Search.

6. The site provides good feedback (e.g. progress indicators or messages) when needed (e.g. during checkout).
7. User confirmation is required before carrying out potentially "dangerous"actions (e.g. deleting something).
8. Confirmation pages are clear.
9. Error messages contain clear instructions on what to do next.
10. When the user needs to choose between different options (such as in a dialog box), the options are obvious.
11. Error messages are written in a non-derisory tone and do not blame the user for the error.
12. Pages load quickly (5 seconds or less).
13. The site provides immediate feedback on user input or actions.
14. Where tool tips are used, they provide useful additional help and do not simply duplicate text in the icon, link or field label.
15. When giving instructions, pages tell users what to do rather than what to avoid doing.
16. The site shows users how to do common tasks where appropriate (e.g. with demonstrations of the site's functionality).
17. The site provides feedback (e.g. "Did you know?") that helps the user learn how to use the site.
18. The site provides context sensitive help.
19. Help is clear and direct and simply expressed in plain English, free from jargon and buzzwords.
20. The site provides clear feedback when a task has been completed successfully.
21. Important instructions remain on the screen while needed, and there are no hasty time outs requiring the user to write down information.
22. Fitts' Law is followed (the distance between controls and the size of the controls is appropriate, with size proportional to distance).
23. There is sufficient space between targets to prevent the user from hitting multiple or incorrect targets.

24. There is a line space of at least 2 pixels between clickable items.
25. The site makes it obvious when and where an error has occurred (e.g. when a form is incomplete, highlighting the missing fields).
26. The site uses appropriate selection methods (e.g. pull-down menus) as an alternative to typing.
27. The site does a good job of preventing the user from making errors.
28. The site prompts the user before correcting erroneous input (e.g. Google's "Did you mean...?").
29. The site ensures that work is not lost (either by the user or site error).
30. Error messages are written in plain language with sufficient explanation of the problem.
31. When relevant, the user can defer fixing errors until later in the task.
32. The site can provide more detail about error messages if required.
33. It is easy to "undo"(or "cancel") and "redo"actions.

8.2 Avaliação da comunidade

Devido as soluções serem open source, uma análise sobre a comunidade em torno da ferramenta é um aspecto importante na avaliação das ferramentas. Como todas as ferramentas possuem seu código disponibilizado dentro do github.com, o próprio site fornece números interessantes sobre os projetos, adicionados a estes números serão avaliados também números da lista de discussão de email e os canais de comunicação do projeto.

Os critérios para avaliar serão:

Watchers - Número de pessoas que acompanham o projeto (novos commits), chamados *watchers*

Forks - Número de versões de terceiros em paralelo do projeto (fork).

Commits - Número de commits no ano de 2010, até 12 de novembro.

Contribuidores - Número de contribuidores do projeto.

Pageviews - Número de pageviews na página do projeto entre agosto - novembro 2010.

Lista de email - Possui lista de discussão por email?

Número pessoas lista - Número de pessoas cadastradas na lista de discussão por email.

Número mensagens lista - Número de mensagens postadas no ano de 2010 na lista de email.

Twitter - Possui Twitter?

Seguidores twitter - Quantos seguidores possui no Twitter?

IRC - Possui canal no IRC (Internet Relay Chat)?

9 Avaliação das ferramentas

Este capítulo se dedica a fazer a avaliação entre as ferramentas Radiant, Refinery, BrowserCMS e Zena, conforme os aspectos apresentados anteriormente.

9.1 Avaliação da Comunidade Open Source

Esta tabela de avaliação se baseia em dados retirados dos repositórios de dados (`http://www.github.com`), lista de discussão de email, e redes sociais como o Twitter (`http://www.twitter.com`)

Tabela 9.1: Tabela comparativa da comunidade open source de cada ferramenta

Critério	Radiant	Refinery	BrowserCMS	Zena
Watchers	1087	897	711	133
Forks	215	219	84	10
Commits	399	3090	99	386
Contribuidores	52	58	17	3
Pageviews	81942	70588	8259	723
Lista de email	Sim	Sim	Sim	Sim
Número pessoas email	409	344	369	70
Número mensagens lista	1716	1653	859	140
Twitter	Sim	Sim	Sim	Sim
Seguidores twitter	276	45	308	47
IRC	Sim	Sim	Sim	Sim

Pode-se observar nesta tabela que o Radiant e Refinery se destacam quanto ao número de pessoas que acompanham o projeto, fazem suas versões do mesmo, contribuem e participam da lista de email. São de fato os dois projetos mais ativos nestes termos. BrowserCMS fica um pouco aquém dos dois e o Zena nota-se que está bem longe dos outros neste aspecto de comparação. É visível que todas as ferramentas possuem variados canais de comunicação com seus usuários (todos os canais avaliados), ressalta a importância dada

à comunidade pelos seus mantenedores.

9.2 Avaliação por checklist

A avaliação pela checklist apresentada no capítulo anterior trouxe resultados interessantes. Pelo fato dela ser muito longa e em formato excel, as checklists de cada ferramenta estão disponíveis nesta url `https://github.com/rafaelsouza/tcc/tree/master/checklists/`.

As imagens com a tabela e o gráfico do resultado da checklist contém dois critérios não utilizados na checklist (Trust & Credibility, Writing & Content Quality), pois está fora do escopo desta avaliação.

Abaixo há o resultado da avaliação de cada ferramenta e os pontos importantes levantados pela inspeção por checklist efetuada.

9.2.1 Radiant

Summary of results

	Raw score	# Questions	# Answers	Score
Home Page	12	20	17	85%
Task Orientation	3	44	31	55%
Navigation & IA	15	29	24	81%
Forms & Data Entry	5	23	17	65%
Trust & Credibility	0	13	0	
Writing & Content Quality	0	23	0	
Page Layout & Visual Design	31	38	35	94%
Search	-20	20	20	0%
Help, Feedback & Error Tolerance	9	37	33	64%
Overall score		247	177	63%

Figura 9.1: Tabela com o resultado da inspeção por checklist no Radiant

Esta tabela foi gerada a partir da checklist e mostra um resultado geral, o qual ressalta a ausência do campo de busca nas funções da ferramenta. Nota-se também um desempenho abaixo da média no critério de Task Orientation.

Porém, salienta-se o bom desempenho nos critérios de Page Layout & Visual Design e Home Page.

A figura 9.2 demonstra o resultado em forma de gráfico para uma melhor visualização.

Alguns pontos que se destacaram durante a inspeção foram as mensagens de confirmação 9.3 e de erro 9.4 , as suas opções para salvar, salvar e continuar editando ou

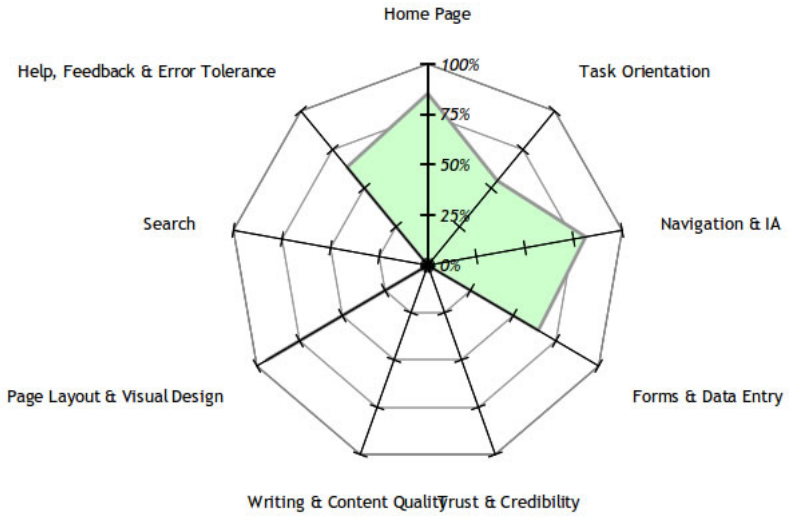

Figura 9.2: Gráfico com o resultado da inspeção por checklist no Radiant

cancelar 9.5.

Pontos que se destacaram negativamente, foram principalmente a falta da funcionalidade de busca 9.6 e a ausência de opções para editar o conteúdo de forma mais robusta e usual, pois o sistema suporta apenas edição direta de html ou filtros como MarkDown e Textile 9.7

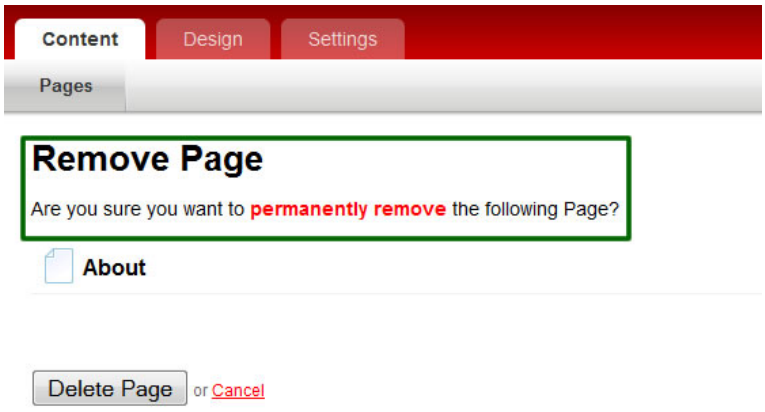

Figura 9.3: Mensagem de confirmação de ação do Radiant

9.2.2 Refinery

A figura 9.8 contendo a tabela gerada a partir da checklist, destaca-se o alto índice nos critérios da Home Page, Page Layout & Visual Design e Task Orientation. Os destaques

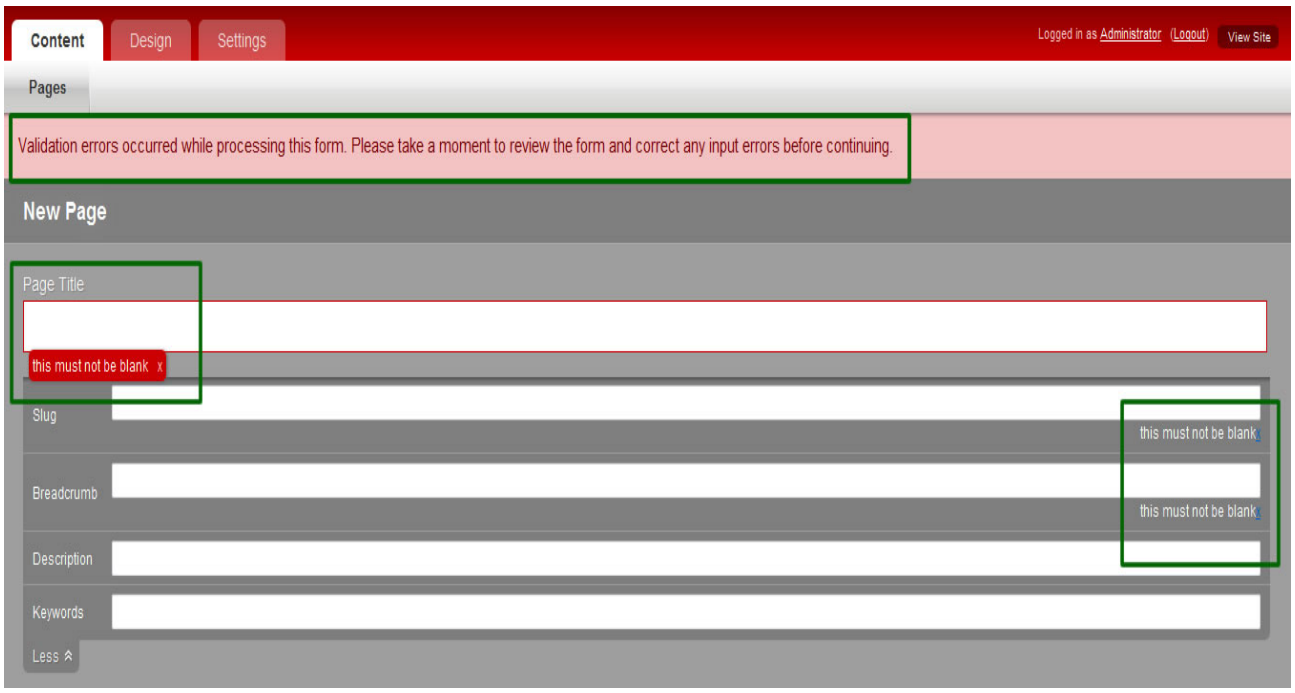

Figura 9.4: Mensagem de erro do Radiant

negativos ficaram nos critérios de Search e de Help,Feedback & Error Tolerance.

A figura 9.9 demonstra o resultado em forma de gráfico para uma melhor visualização.

Os pontos que se destacaram durante a inspeção da ferramenta foram a presença dos atalhos na página principal do Refinery 9.10 e as tooltips presentes em vários labels do mesmo 9.11

Pontos que se destacaram negativamente foram a funcionalidade de busca, embora presente, é uma busca simples sem maiores detalhes. A figura 9.12 mostra a ausência de uma indicação clara de onde o usuário se encontra, se está editando ou criando um conteúdo. A figura 9.13 ilustra a falta de distinção entre campos obrigatórios e não obrigatórios. No caso o campo title é obrigatório e não vem com indicação de sua obrigatoriedade.

9.2.3 BrowserCMS

Esta figura 9.14 contendo a tabela gerada a partir da checklist, destaca-se o alto índice no critérios de Navigation IA.

Os destaques negativos ficaram nos critérios da Search,Forms & Data En-

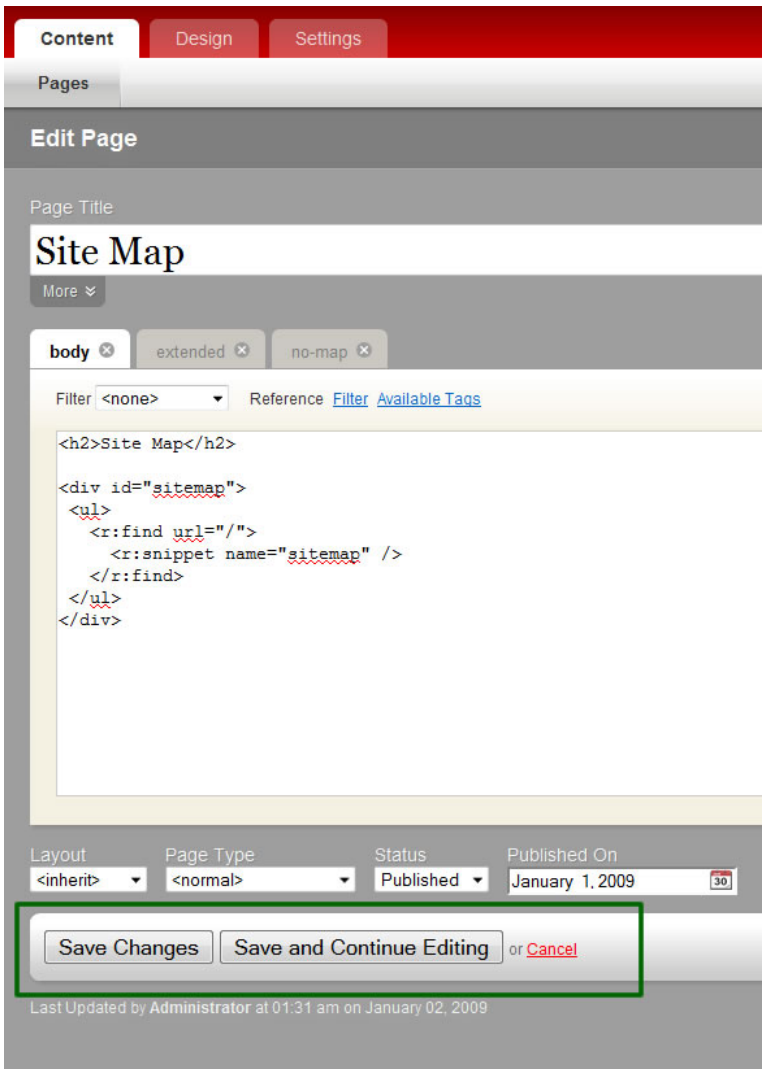

Figura 9.5: Opções para salvar conteúdo no Radiant

try,Help Feedback & Error Tolerance.

A figura 9.15 demonstra o resultado em forma de gráfico para uma melhor visualização.

O BrowserCMS traz uma proposta de interface diferente de Radiant e Refinery, onde o usuário edita o conteúdo fazendo diretamente uma pré visualização do mesmo. Há uma curva de aprendizado para a ferramenta, mas após acostumado ela traz alguns benefícios como a possibilidade de reutilizar conteúdos de outras páginas, facilitando a manutenção do conteúdo.

Um ponto a se destacar, está na figura 9.16 onde ilustra a possibilidade de editar o conteúdo diretamente da pré-visualização da página. Um ponto negativo desta imagem é a falta de explicação dos ícones utilizados, tornando o entendimento dos mesmos mais complicado.

Figura 9.6: Falta de busca no radiant

Pontos que se destacaram negativamente foram a falta de opção para cancelar uma ação conforme a figura 9.17 mostra. Outro problema apontado na figura 9.18 foi a mensagem de confirmação das ações executadas que aparecem muito rapidamente e estão mal localizadas, dando a impressão que o usuário não completou determinada ação.

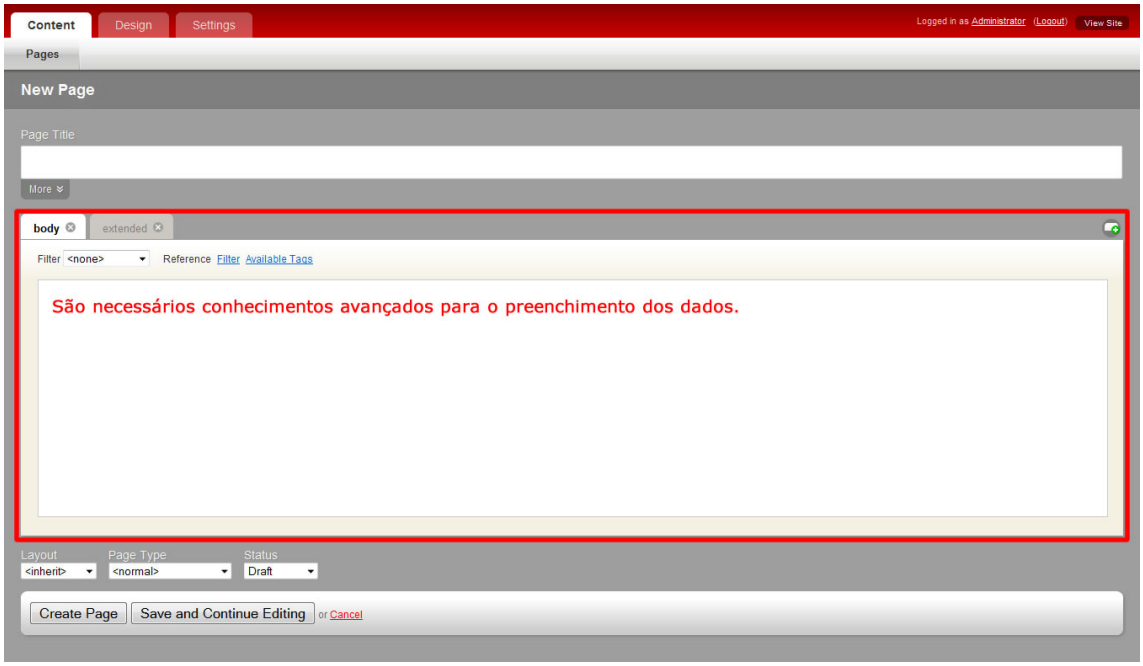

Figura 9.7: Ausência de um editor de html mais robusto para a criação de conteúdo

Summary of results

	Raw score	# Questions	# Answers	Score
Home Page	16	20	16	100%
Task Orientation	20	44	29	84%
Navigation & IA	14	29	24	79%
Forms & Data Entry	8	23	19	71%
Trust & Credibility	0	13	0	
Writing & Content Quality	0	23	0	
Page Layout & Visual Design	33	38	35	97%
Search	-2	20	20	45%
Help, Feedback & Error Tolerance	13	37	34	69%
Overall score		247	177	78%

Figura 9.8: Tabela com o resultado da inspeção por checklist no Refinery

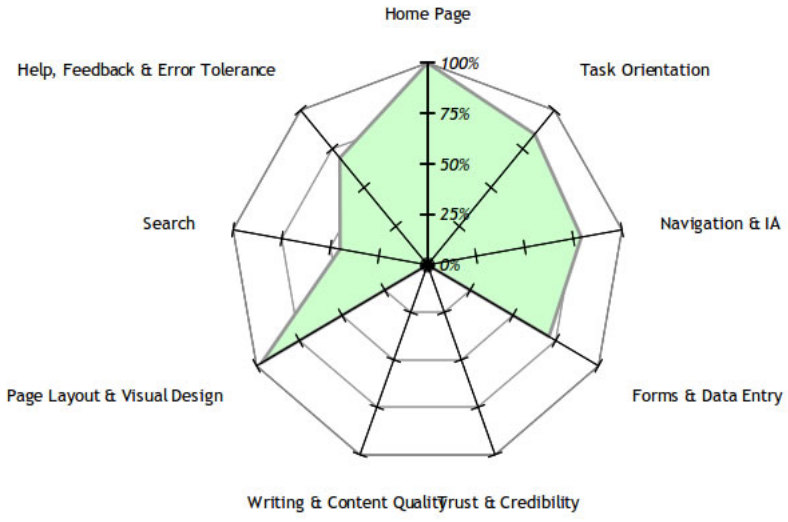

Figura 9.9: Gráfico com o resultado da inspeção por checklist no Refinery

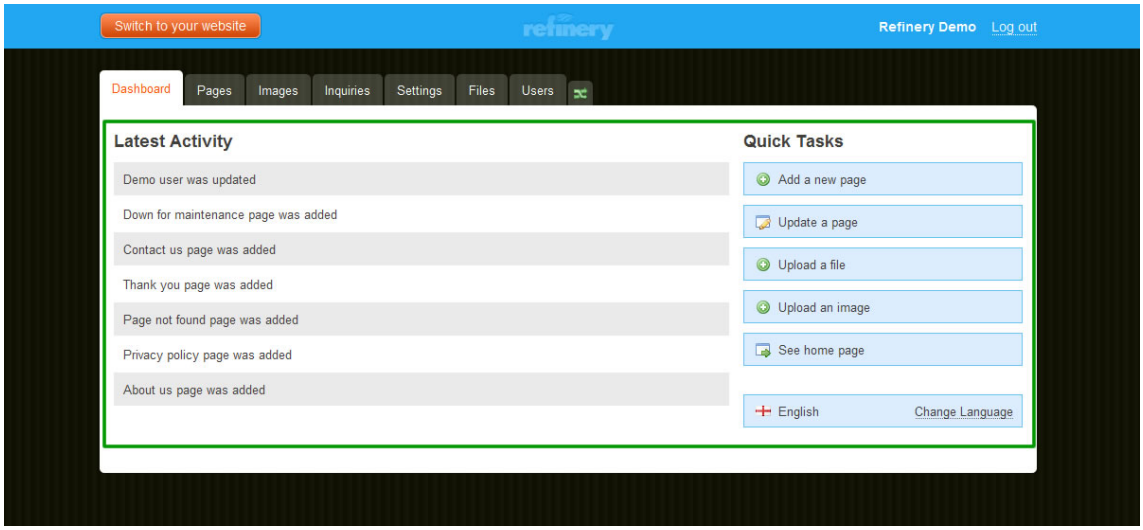

Figura 9.10: Painel com acesso rápido as funções do Refinery

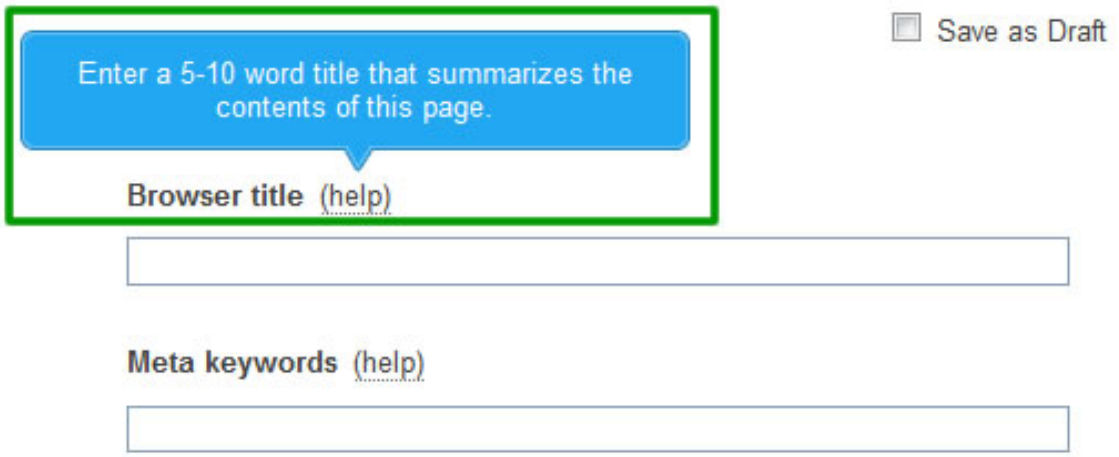

Figura 9.11: Tooltip de ajuda no Refinery

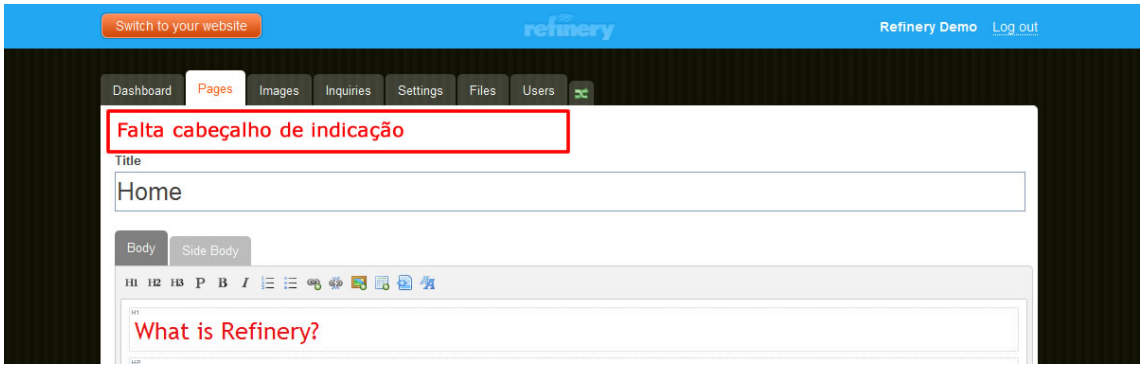

Figura 9.12: Falta indicação clara de onde o usuário se encontra no Refinery

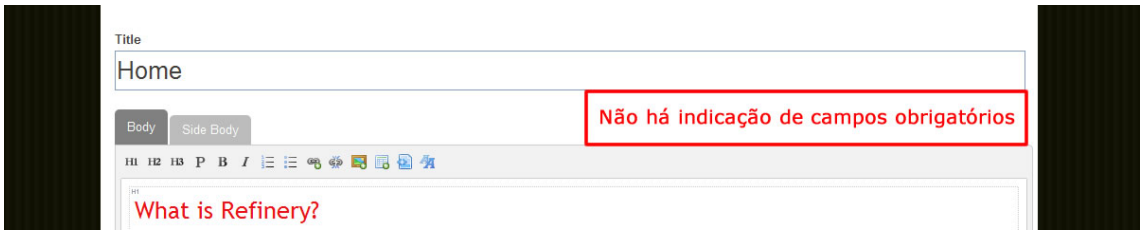

Figura 9.13: Falta uma indicação de campo obrigatório no Refinery.

Summary of results

	Raw score	# Questions	# Answers	Score
Home Page	10	20	17	79%
Task Orientation	14	44	33	71%
Navigation & IA	19	29	25	88%
Forms & Data Entry	3	23	17	59%
Trust & Credibility	0	13	0	
Writing & Content Quality	0	23	0	
Page Layout & Visual Design	19	38	35	77%
Search	2	20	19	55%
Help, Feedback & Error Tolerance	4	37	34	56%
Overall score		247	180	69%

Figura 9.14: Tabela com o resultado da inspeção por checklist no BrowserCMS

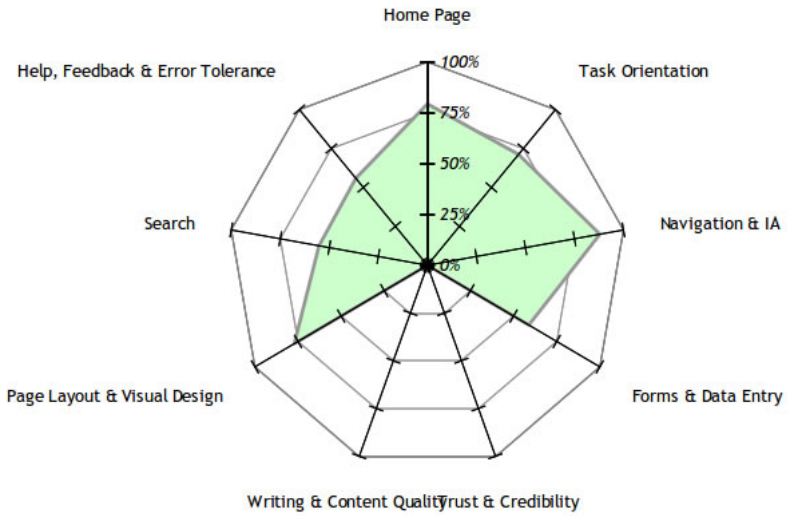

Figura 9.15: Gráfico com o resultado da inspeção por checklist no BrowserCMS

Figura 9.16: Edição de conteúdo em modo de pré visualização do BrowserCMS.

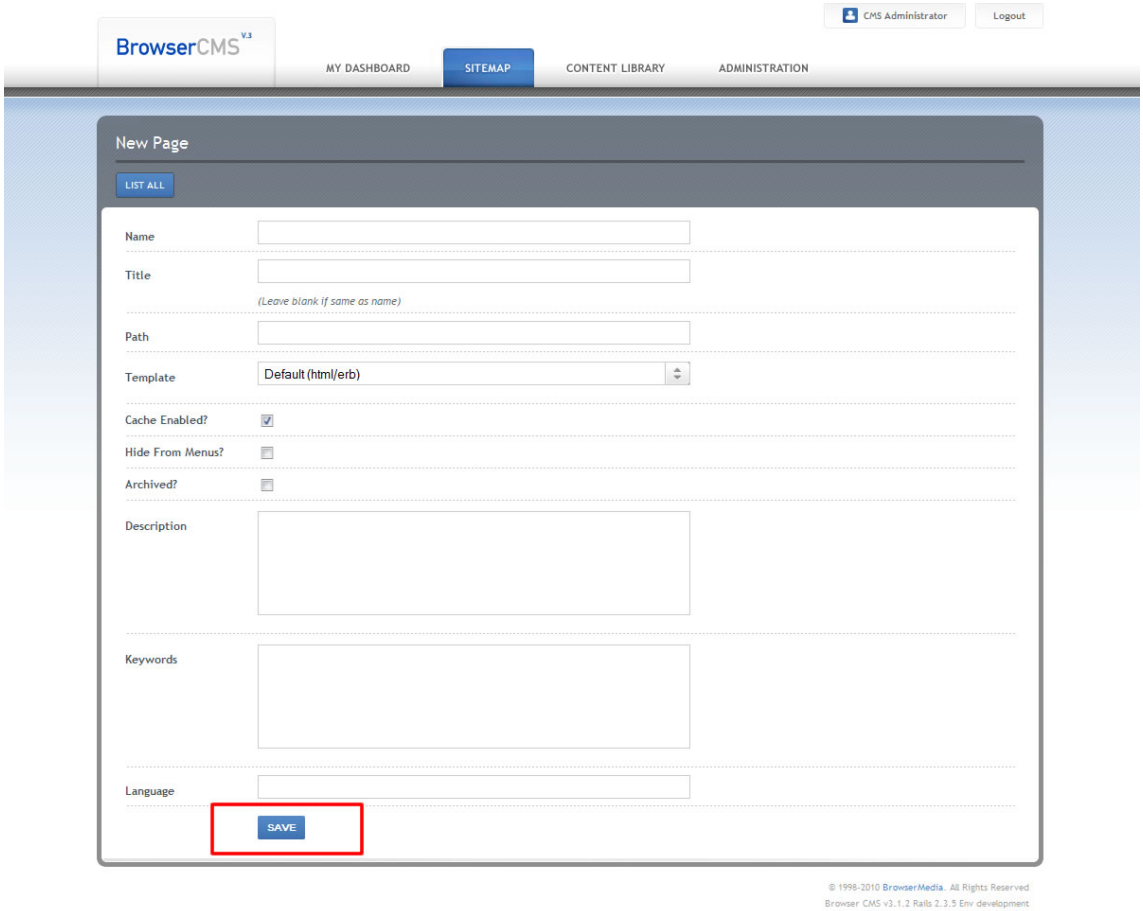

Figura 9.17: Ausência de botão para cancelar ação no BrowserCMS

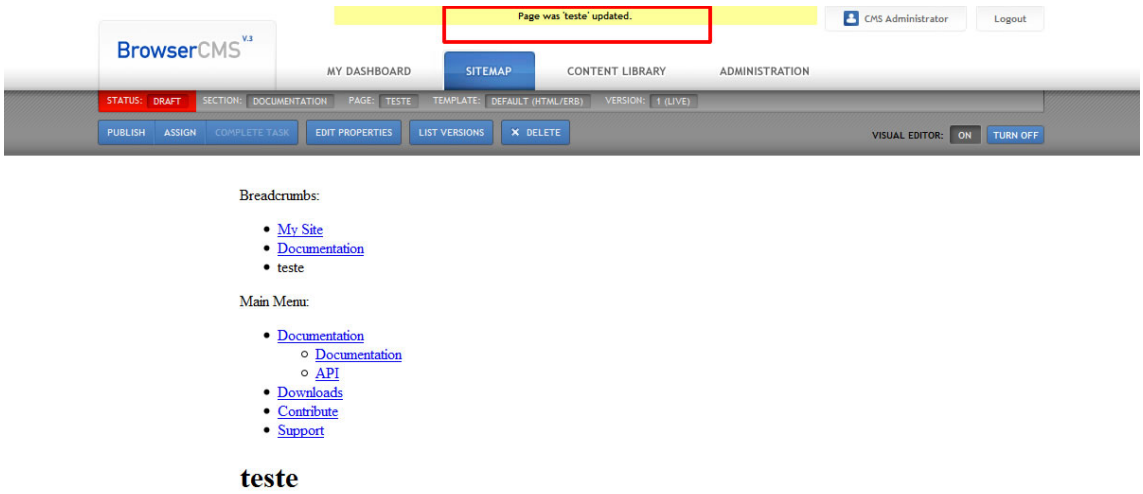

Figura 9.18: Mensagem de confirmação de ação aparece rapidamente e mal localizada no BrowserCMS

10 Conclusão

Este trabalho apresentou uma avaliação entre quatro ferramentas CMS de código aberto desenvolvidas em Ruby on Rails. Foram escolhidos dois aspectos, usabilidade e a comunidade open source, como medidas para avaliar as ferramentas. O processo de escolha dos aspectos foi complexo, pois definir o que medir entre as ferramentas para guiar a avaliação é bastante complexo, visto que, cada CMS possui suas peculiaridades.

As avaliações apontaram o Refinery com uma vantagem em relação às outras ferramentas, principalmente no aspecto da usabilidade. Embora tenha recebido uma avaliação melhor, a conclusão sobre uma escolha de uma ferramenta CMS deve ser levado em conta principalmente o contexto ao qual a ferramenta irá resolver o problema. Dependendo do contexto, outra ferramenta pode ser mais adequada à necessidade da aplicação.

Durante o trabalho uma descoberta interessante foi o fato de não haver uma inspeção de checklist específica para aplicações web, inclusive é uma sugestão para trabalhos futuros, elaborar uma checklist voltada para aplicações web. Outros possíveis trabalhos futuros seriam a extensão da pesquisa para atingir mais aspectos e ferramentas, incluindo outras plataformas de desenvolvimento, trazendo aspectos como performance, cobertura de testes da ferramenta e análise da arquitetura das mesmas.

Referências Bibliográficas

Apostila de engenharia de usabilidade. http://www.labiutil.inf.ufsc.br/hiperdocumento/.

Usability inspection methods, a. http://www.sigchi.org/chi95/proceedings/tutors/jn_bdy.htm.

Engenharia de usabilidade, b. http://www.labiutil.inf.ufsc.br/hiperdocumento/Engenharia_de_Usabilidade_Nielsen.doc.

Radiant official website. http://radiantcms.org/.

Maglev official website. http://rubini.us/.

What is a cms? http://www.cmscritic.com/what-is-a-cms/.

O que é cms?, 2007. http://www.navita.com.br/portal/newsroom/definicoes/cms.html.

Bob Boiko. *Content Management Bible, 2nd edition*. Wiley Publishing, 2005.

M. De Marsico C. Ardito, M. F. Costabile. An approach to usability evaluation of e-learning applications, 2002. http://www.springerlink.com/index/755507r7144m3845.pdf.

Michael Fitzgerald. *Ruby - Pocket Reference*. O'Reilly, 2007a.

Michael Fitzgerald. *Ruby - Pocket Reference*. O'Reilly, 2007b.

David Flanagan and Yukihiro Matsumoto. *The Ruby Programming Language*. O'Reilly, 2008.

Andreas Holzinger. Usability engineering methods for software developers, 2005. http://www.sigchi.org/chi95/proceedings/tutors/jn_bdy.htm.

Ulrich Kampffmeyer. Ecm enterprise content management, 2006.

MARTIN MAGUIRE. Context of use within usability activities, 2001. http://www.cse.chalmers.se/research/group/idc/ituniv/kurser/05/ucd/papers/Maguire%202001.pdf.

Nirav Mehta. *Choosing an Open Source CMS*. Packt Publishing, 2009.

ruby.gemstone.com. Maglev official website, 2009. http://ruby.gemstone.com/.

Bruce Stewart. An interview with the creator of ruby, 2001. http://linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html.

www.ruby lang.org, 2009. http://www.ruby-lang.org/.