CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

QUALIDADE DE CÓDIGO CSS: UMA PROPOSTA DE MÉTRICA DA MANUTENIBILIDADE DE CÓDIGO CSS

VICTOR CARNEIRO SALVADOR

Orientador: Prof. Flávio Roberto dos Santos Coutinho
Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

BELO HORIZONTE
JULHO DE 2015

VICTOR CARNEIRO SALVADOR

QUALIDADE DE CÓDIGO CSS: UMA PROPOSTA DE MÉTRICA DA MANUTENIBILIDADE DE CÓDIGO CSS

BELO HORIZONTE
JULHO DE 2015

Sumário

1 Introdução

A *world wide web*, originalmente proposta como um meio para compartilhamento de documentos por Tim Berners-Lee, torna-se cada vez mais popular, tendo passado a ser usada para a criação de páginas mais complexas e até mesmo de sistemas de informação, como comércio eletrônico, fóruns, clientes de email, portais de compartilhamento de vídeo etc.

Inicialmente proposto por Håkon Wium Lie e Bert Bos, a linguagem *Cascading Style Sheet* (CSS) propunha a separação do documento de conteúdo — o arquivo HTML — da apresentação das páginas *web*. Sendo um dos três padrões fundamentais da W3C[1] para desenvolvimento de conteúdo *web*, juntamente com o HTML e o Javascript, o CSS se tornou largamente utilizado para definir a aparência e até mesmo certos comportamentos interativos em páginas *web*.

Apesar das vantagens trazidas pela separação de responsabilidades, passou-se a gerar código CSS complexo e de manutenibilidade onerosa (MESBAH; MIRSHOKRAIE, 2012). Escrever regras CSS não é uma tarefa trivial, visto que as características da linguagem, como herança e especificidade de seletores, colocam os desenvolvedores constantemente em situações nas quais se questionam a melhor prática para se definir as propriedades utilizadas. Essas características podem prejudicar o que Keller e Nussbaumer (2010) definem como efetividade e eficiência de código CSS:

- **Efetividade do código:** a folha de estilo é efetiva se o documento de conteúdo ao qual ele é aplicado renderiza da forma desejada.
- **Eficiência do código:** folhas de estilo que causam o mesmo efeito em um documento de conteúdo ainda pode diferir significativamente no modo em que ela aplica a associação de propriedade. Maximizar a eficiência do código CSS significa aplicar a associação de propriedades de uma forma que o esforço da autoria, manutenção e eventual reutilização seja minimizado.

Essas característica da linguagem CSS, são muitas vezes ignoradas pelos codificadores, ou pelos desenhistas, que definem o estilo de uma página *web*. Esse descaso pode gerar um número alto de conflitos, isto é, ao se modificar uma propriedade, de um elemento qualquer, pode-se gerar um efeito indesejado. Estes conflitos podem ser vistos como efeitos colaterais, e podem se apresentar em um grande número de cenários e, por isso, são muito difíceis de se identificar em um código pronto.

[1] World Wide Web Consortium - http://www.w3.org/

1.1 Justificativa

Alterações de dimensão, ou alterações de visibilidade dos elementos podem causar efeitos indesejados na renderização de elementos vizinhos, parentes ou filhos. Esses efeitos colaterais são muito comuns em edição de páginas onde a definição de estilo está distribuída em vários arquivos, ou em um único grande arquivo CSS centralizado.

Pode-se notar, então, a dificuldade de se manter um código CSS sem falhas durante a construção de uma página web. Portanto, existe a necessidade de se manter um alto grau de manutenibilidade. A manutenibilidade de um sistema é definida como a facilidade com a qual um software, ou componente, pode ser modificado para corrigir falhas, melhorar performance, ou adaptar-se à mudança de ambiente (IEEE, 1990). A partir dessa definição, pode-se identificar uma medida de manutenibilidade para códigos CSS, considerando-se que onde houver alta complexidade haverá a necessidade de se manter o funcionamento, ou adaptação, da apresentação do documento.

A manutenção e modificação de *software* são etapas essenciais para o seu tempo de vida, e isso não é diferente para aplicações *web*. Sendo uma tarefa essencial, e complexa, entende-se que seja necessário encontrar uma forma de mitigar os possíveis impactos na modificação, ou evolução, das folhas de estilo dos projetos web.

As linguagens de folha de estilo, como o CSS, são muito pouco documentadas (MARDEN; MUNSON, 1999; QUINT; VATTON, 2007; GENEVES et al., 2012) e, como identificado por Mesbah e Mirshokraie (2012), analisar código CSS com uma perspectiva de manutenção ainda não foi explorada em nenhum trabalho científico. Portanto, há necessidade de se definir a qualidade do código CSS, com objetivo de se manter um nível de manutenibilidade da apresentação de páginas web. Para medir este nível, será feita neste trabalho, uma proposta de métrica de qualidade de código CSS, focando a manutenção do código.

1.2 Objetivos

Esta pesquisa tem por objetivo propor uma métrica de manutenibilidade de código CSS. Para tanto, será feito um levantamento das propriedades da linguagem que modificam a facilidade da manutenção de seu produto, considerando a visão do autor e os aspectos funcionais.

Pretende-se identificar, junto à comunidade desenvolvedora de CSS, os parâmetros fundamentais da qualidade de código CSS, bem como as principais dificuldades na etapa de manutenção de uma folha de estilo. Utilizando-se desses parâmetros, pretende-se criar um modelo de métrica para identificarmos quantitativamente a manutenibilidade de um código fonte, validando com uma verificação cruzada da quantidade de retrabalho, referente ao estilo, com o valor obtido pela métrica.

2 Fundamentação Teórica

2.1 HTML

HTML é a linguagem principal para criação de documentos e aplicações na *Web* para o uso de todos, em qualquer lugar (W3C, 2015b).

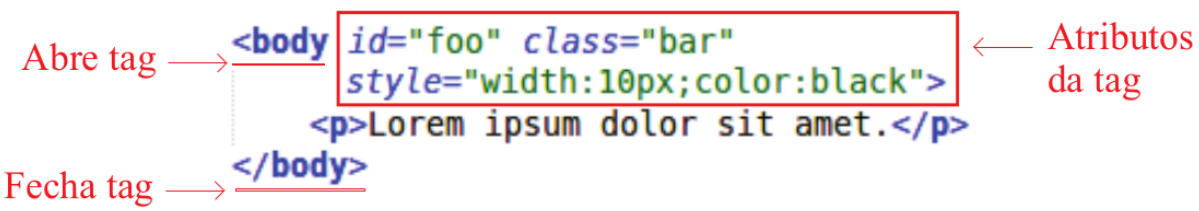
O documento HTML consiste em uma árvore de elementos e texto. Cada elemento é representado por uma *tag* de abertura e uma de fechamento. As *tags* têm de estar todas aninhadas completamente, sem haver sobreposição. Os elementos podem ter atributos que controlam o seu comportamento (HICKSON et al., 2014). Na Figura 1 está representada a estrutura básica de um documento HTML.

Figura 1 – Exemplo de arquivo HTML válido.

```
<!DOCTYPE html>
<html>
    <body>
        <h1 id="foo-header" class="header">Foo</h1>

        <p title="foo">
        Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
        tempor incididunt ut labore et dolore magna aliqua.
        </p>

        <div id="div-bar" style="background-color:black; color:white;">
            <h2 class="header">Bar</h2>

            <p title="lorem">
            Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
            tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
            quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
            consequat.
            </p>
        </div>
    </body>
</html>
```

Fonte: próprio autor

Figura 2 – Estrutura de uma *tag*.



Fonte: próprio autor

Uma *tag* HTML, pode possuir atributos que podem definir uma meta informação, com o objetivo de organizar o arquivo HTML, ou definir seu estilo. Esses atributos devem sempre ser definidos na *tag* de abertura, e são representados por um par chave/valor. Podemos ver na Figura 2 uma estrutura simples de um *tag* com os atributos *id*, *class* e *style* definidos.

Pode-se notar na Figura 1 a utilização dos atributos *id*, *title* e *style*, que representam a identificação única do elemento, uma meta informação identificando a sua utilidade e o estilo aplicado a ele, respectivamente.

Dentro do documento HTML pode-se utilizar as *tags* <style/> e <script/>, que definem escopos de estilo e linguagens de *script*, como pode-se observar na Figura 3.

Figura 3 – Exemplo de arquivo HTML válido.

```
<style type="text/css">
    body{
        background-color:yellow;
    }
    p {
        color:blue;
    }
</style>
```

```
<script type="text/javascript">
    function myFunction() {
        document.getElementById("foo-header").innerHTML = "Hello JavaScript!";
    }
</script>
```

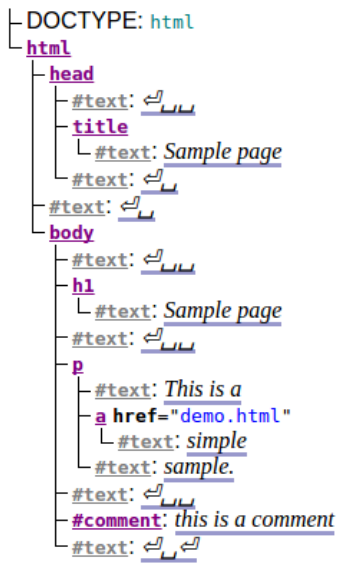
Fonte: próprio autor

É uma boa prática de desenvolvimento *web* manter as estruturas em arquivos diferentes, evitando ao máximo a utilização das *tags* de escopo. Essa separação mantém uma organização do código, possibilita o reuso em outras páginas, ou até mesmo em outros sistemas, e também melhoram o desempenho, pois podem ser mantidos em cache, diminuindo a carga de dados necessária para renderização de um página *web*. Porém, esta prática é difícil de gerenciar, dificultando a identificação de erros e o reuso de seus componentes.

Os navegadores *web* (*Browsers*) traduzem esse formato em uma árvore DOM (*Document Object Model*). Uma árvore DOM é uma representação em memória de um documento, que possui vários nós, sendo que cada nó contém um elemento ou trecho de texto do documento.

Na Figura 4, vê-se que o elemento raiz da árvore é o html, que é sempre o primeiro elemento de um documento e que contém todos os outros. Cada elemento do HTML é representado por um nó, onde todos os nós que se encontram nos níveis abaixo deste são denominados descendentes (*descendant*). Dentro da árvore DOM, os nós que se encontram exatamente um nível abaixo são os filhos (*child*) e os nós que se encontram no mesmo nível são chamados de irmãos (*sinbling*). Os nós denominados de *text* são os que encapsulam os textos inseridos dentro

Figura 4 – Exemplo de estrutura da árvore do DOM



Fonte: Hickson et al. (2014)

dos elementos HTML, então estes serão os nós que existem em maior número no DOM.

A árvore DOM é utilizada para localização dos nós do HTML, as linguagens CSS e javascript, utilizam da estrutura do DOM para encontrar os elementos e identificar quais são os elementos afetados por suas diretivas. O navegador determina a partir dos nós selecionados pelos seletores CSS, quais serão os elementos afetados pela regra.

2.2 CSS

CSS é um mecanismo simples para adicionar estilo (*e.g.*, fontes, cores, espaçamento) em documentos *Web* (W3C, 2015a).

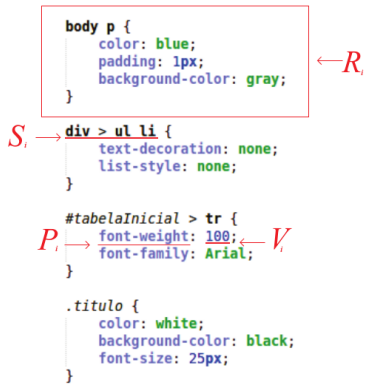
Uma folha de estilo C pode ser vista como um conjunto de regras R, composto por regras simples R_i, cada uma composta por um seletor S_i e um conjunto de pares: propriedade P_i e seus valores V_i. Os seletores definem a quais elementos de um documento, serão aplicadas as propriedades definidas pela regra à qual elas pertencem (GENEVES et al., 2012).

A Figura 5 ilustra uma folha de estilo, com as regras representadas pelo conjunto de seletores, que precedem as chaves, e os pares "propriedade/valor"que as compões, dentro das chaves.

2.2.1 Seletores

Um seletor é uma cadeia de uma, ou mais, sequências de seletores simples, separados por combinadores. Os seletores simples são cadeias de caracteres que representam um elemento

Figura 5 – Exemplo de uma folha de estilo C.



Fonte: próprio autor

do html: o seletor universal, representado pelo simbolo $*$, indica que a regra será aplicada a todos os elementos que estejam no DOM. O seletor de elementos HTML é representado pelo nome da *tag* de um elemento, por exemplo $h1$. O seletor de classe, que seleciona todos os elementos que possuam o atributo $class$ especificado pelo seletor, é utilizado escrevendo-se o nome da classe, precedido de um ponto final (.). O seletor de id, que seleciona o elemento do html que possua aquele id, é utilizado escrevendo-se o identificador precedido pelo simbolo $\#$. Simplificando, sem perder generalidade, pode-se considerar que regras são feitas de seletores únicos que definem uma única propriedade por vez. Os seletores S_i, chamados de padrões na especificação do CSS (ÇELIK et al., 2009), definem uma função booleana na forma:

$$expression * element \rightarrow boolean \tag{1}$$

que define se um elemento é, ou não, selecionado pela expressão do seletor.

Os combinadores são propriedades que definem relações entre os elementos de um documento. Existem três formas de combinadores: descendentes, filhos e irmãos. O combinador de descendente descreve qualquer elemento que esteja um nível abaixo na árvore DOM, e são representados pelo espaço em branco, *e.g.* "$body\ p$". O combinador de filho descreve os elementos que estão exatamente um nível abaixo do nó, este combinador é representado pelo sinal de maior ($>$), *e.g.* "$body > p$". O combinador de irmãos descreve os elementos que estão no mesmo nível da árvore, existem duas variações, uma para o próximo irmão adjacente ($+$) e um para todos os irmãos (~).

Uma pseudo classe, é um elemento de seleção que especifica estado ou localização do elemento. Por exemplo, a pseudo classe ":nth-first-child(n)"identifica o n-ésimo elemento filho contando a partir do primeiro, podendo assim ser classificado como uma pseudo classe de localização. A pseudo classe ":hover"identifica um elemento que esteja sob o cursor do apontador (*mouse*), sendo assim uma pseudo classe de estado. Ainda existem os seletores de atributos, que selecionam elementos que possuam determinados atributos, permitindo a utilização de

expressões para seleções parciais, *i.e.*, atributos que comecem, possuam ou terminem com uma cadeia de caracteres específicos.

Regras simples como as demonstradas na Figura 5 são fáceis de se criar, mas quando utilizados combinadores e pseudo classes, a complexidade da autoria escala. Além da complexidade dos seletores, pode-se apontar o efeito cascata, gerado pelo mecanismo de precedência e aplicação das propriedades aos elementos HTML.

2.2.2 Efeito Cascata

O efeito cascata do CSS se dá devido à ordem de precedência dos valores de propriedades definidas para cada elemento. O mecanismo de renderização do navegador recebe uma lista desordenada dessas propriedades, e as organiza pela precedência das declarações delas. Essa ordem é definida de acordo com os critérios listados a seguir, em ordem decrescente de prioridade (FANTASAI; ATIKNS, 2015).

- **Origem e Importância:** Cada regra de estilo possui uma origem, que define onde ela estará na cascata, e a importância se refere à utilização, ou não, de um atributo que a explicita.
- **Escopo:** Uma declaração pode ter uma subárvore do DOM como escopo, afetando somente os elementos pertencentes a esta subárvore. Para declarações normais, o escopo mais interno tem prioridade, para as regras definidas como importantes as do escopo mais externo sobrescreverão.
- **Especificidade:** O calculo de especificidade conta a ocorrência de seletores de ID, classe e tipo (*tags* e *pseudo-elements*), e faz-se uma soma ponderada dessas ocorrências. A declaração com maior especificidade tem prioridade.
- **Ordem de aparição:** A última declaração no documento tem a maior prioridade. Isto significa que a localidade será levada em conta, para isso, considera-se que as folhas de estilo são concatenadas ao documento na ordem em que são declaradas.

Uma propriedade pode ter sua importância declarada explicitamente através do valor **!important**, que sobrescreverá todas as propriedades que possuírem maior prioridade. Se duas propriedades possuírem o valor **!important**, as outras especificações de prioridade terão efeito.

Outra propriedade do CSS que determina o funcionamento em cascata da aplicação de estilo é a herança. Cada propriedade de estilo possui um valor padrão de herança, que indica se aquela propriedade é propagada para seus filhos, essa herança pode ser definida explicitamente, a partir do valor da propriedade **inherit**.

2.3 Qualidade de *Software*

Com a finalidade de propor uma métrica, será necessário criar um arcabouço teórico sobre as técnicas e medições de qualidade de *software* e código fonte.

A qualidade de *software* faz um estudo sobre o produto do código fonte, considerando fatores produtivos e algumas vezes subjetivos. Em Pressman (2010), destacam-se os fatores de qualidade de *software*, definidos pela ISO 9126, apresentados a seguir:

- **Funcionalidade**. Grau em que o *software* satisfaz as necessidade declaradas.
- **Confiabilidade**. Período de tempo em que o *software* está disponível para uso.
- **Usabilidade**. Grau em que o *software* é fácil de usar.
- **Eficiência**. Grau em que o *software* faz uso otimizado dos recursos do sistema.
- **Manutenibilidade**. Facilidade com a qual podem ser feitos reparos no *software*.
- **Portabilidade**. Facilidade com a qual o *software* pode ser transposto de um ambiente para outro.

Essas seis características chave apresentam a qualidade do produto de *software*, que são difíceis de se medir quantitativamente e dependem da análise subjetiva de um especialista. Essa subjetividade torna as métricas difíceis de se reproduzir, portanto, é quase impossível determinar uma relação entre estados diferentes do produto.

Whitmire (1997) define a qualidade de *software* orientado à objetos (OO) a partir de nove características distintas e mensuráveis de projetos OO:

- **Tamanho**, definido em termos de quatro perspectivas: população, volume, comprimento e funcionalidade. População é medida pela contagem estática das entidades OO, tais como classes ou operações. Medidas de volume são medidas de população coletadas dinamicamente em função de um determinado instante de tempo. Comprimento é a medida de uma cadeia de elementos de projeto interconectadas, *e.g*, a profundidade de uma árvore de herança. Métricas de funcionalidade fornecem uma indicação indireta do valor entregue ao cliente.
- **Complexidade** é determinada pela forma com a qual as classes OO de um projeto se inter-relacionam umas com as outras.
- **Acoplamento**, é definido pelas conexões físicas entre elementos de um projeto OO, *e.g*, o número de colaborações entre as classes ou o número de mensagens passadas entre objetos.

- **Suficiência** é o grau das características exigidas de uma abstração ou o grau das características que um componente de projeto possui na sua abstração, do ponto de vista da aplicação corrente. Ou seja, um componente de *software* é suficiente se reflete plenamente todas as propriedades do objeto de domínio de aplicação que representa.
- **Completeza** se diferencia de suficiência pelo conjunto de características com o qual se compara a abstração ou o componente de projeto. A completeza considera múltiplos pontos de vista da aplicação corrente. Como este critério considera diferentes pontos de vista, tem implicação direta no grau de reusabilidade da abstração.
- **Coesão** é determinada pelo grau em que o conjunto de propriedades que a abstração possui é parte do problema ou do domínio do projeto.
- **Primitividade** é o grau em que uma operação é atômica — *i.e.*, a operação não pode ser construída a partir de uma sequência de outras operações contidas na classe.
- **Similaridade** é o grau em que duas ou mais classes são semelhantes, em termos de sua estrutura, função ou finalidade.
- **Volatilidade** mede a probabilidade de que uma modificação venha a ocorrer em um componente de projeto OO.

Segundo Pressman (2010), as métricas de qualidade de código-fonte também podem ser analisadas em nível de componente. Essas métricas são a de coesão, acoplamento e complexidade.

2.3.1 Métricas de coesão

Bieman e Ott (1994) definem uma coleção de métricas que fornecem indicação da coesividade de um módulo. As métricas são definidas especificando cinco conceitos e medidas.

- **Fatia de dados (*data slice*)** é um caminho retroativo ao longo de um módulo, que procura valores de dados que afetam a posição no módulo em que o caminho teve inicio.
- **Fichas de dados (*data tokens*)** são as variáveis especificadas para um módulo.
- **Fichas aglutinadas (*glue tokens*)** um conjunto de *data tokens* que está contido em um, ou mais, *data slices*.
- **Fichas superaglutinadas (*superglue tokens*)** são os *data tokens* que estão contidos em todos os *data slices*.
- A **aglutinação (*stickiness*)** relativa de uma ficha aglutinante é diretamente proporcional ao número de *data slices* que ela aglutina.

As métricas de coesão definidas por Bieman e Ott (1994) são de três tipos, coesão funcional forte (*strong functional cohesion* — SFC), coesão funcional fraca (*weak function cohesion* — WFC) e adesividade (*adhesiveness*). Todas essas métricas de coesão variam de 0 a 1. Têm valor 0 quando um procedimento tem mais de uma saída e não exibe nenhum dos atributos de coesão indicados por uma métrica particular. Quando não há *superglue tokens*, nenhuma ficha é comum a todos os *data slices*, tem coesão funcional forte igual a zero. Quando não existem *data tokens* comuns a mais de um *data slice*, e o procedimento possui mais de um *data slice*, exibe coesão funcional fraca zero e adesividade zero.

2.3.2 Métricas de acoplamento

O acoplamento de módulos fornece a indicação da conectividade de um módulo a outros módulos, dados globais e ambiente exterior. A métrica para acoplamento de módulos proposta por Dhama (1995) que engloba acoplamento de dados e de fluxo de controle, acoplamento global e acoplamento ambiental. As medidas necessária para calcular acoplamento de módulos são definidas em termos de cada um dos três tipos de acoplamento mencionados. Para acoplamento de dados e de fluxo de controle,

d_i = número de parâmetros de dados de entrada

c_i = número de parâmetros de controle de entrada

d_o = número de parâmetros de dados de saída

c_o = número de parâmetros de controle de saída

Para acoplamento global,

g_d = número de variáveis globais usadas como dados

g_c = número de variáveis globais usadas como controle

Para acoplamento ambiental,

w = número de módulos chamados (*fan-out*)

r = número de módulos que chamam o módulo sendo considerado (*fan-in*)

Usando essas medidas, um indicador de acoplamento de módulo, m_c, é definido do seguinte modo:

$$m_c = \frac{k}{M} \tag{2}$$

em que k é uma constante de proporcionalidade e

$$M = d_i + (a \times c_i) + d_o + (b \times c_o) + g_d + (c \times g_c) + w + r \tag{3}$$

Os valores para k, a, b e c devem ser derivados empiricamente. À medida que o valor de m_c, aumenta, o acoplamento global do módulo diminui. A fim de ter a métrica de acoplamento aumentando conforme o grau de acoplamento aumenta, uma métrica de acoplamento revisada pode ser definida como

$$C = 1 - m_c \tag{4}$$

em que o grau de acoplamento aumenta à medida que as medidas da Equação (3) aumentam.

2.3.3 Métricas de complexidade

Diversas métricas de *software* podem ser calculadas para determinar a complexidade do fluxo de controle do programa. Muitas delas são baseadas no diagrama de fluxo. As métricas de qualidade podem ser usadas para prever informação crítica sobre confiabilidade e manutenibilidade de sistemas de *software* a partir da análise automática do código-fonte, ou informação do projeto procedimental. Métricas de complexidade também fornecem realimentação durante o projeto de *software* para ajudar a controlar a atividade de projeto (PRESSMAN, 2010). Fornecendo informações detalhadas sobre os módulos de *software*, durante o teste e manutenção, ajudando a localizar áreas de potencial instabilidade.

A métrica de complexidade mais amplamente usada (e debatida) para *software* de computador é a complexidade ciclomática, desenvolvida por McCabe e Butler (1989).

3 Trabalhos Relacionados

3.1 Qualidade de Código Clássica

A norma ISO 9126 define seis atributos-chave de qualidade de software de computador, um dos atributos é a manutenibilidade (PRESSMAN, 2010). E a Ieee (1990) define métrica como uma medida quantitativa do grau em que um sistema, componente ou processo possui um determinado atributo. Pode-se definir a medida quantitativa do grau de manutenibilidade de um código fonte como uma métrica.

Existem vários trabalhos na área de medição de software, enquanto Whitmire (1997) discute os atributos chave que definem a qualidade de um sistema de *software*, trabalhos como os de McCabe e Butler (1989), Zuse (1991), Bieman e Ott (1994), Dhama (1995), Zuse (1997) exploram as medições de coesão, acoplamento e complexidade que compõem os nove atributos chave.

Riaz et al. (2009) discutem os trabalhos relacionados à métricas e previsão de manutenibilidade de código fonte. Através de uma revisão bibliográfica, concluem que os modelos de previsão de manutenibilidade mais utilizados se baseiam em técnicas algorítmicas, sem encontrar distinção de qual modelo deve ser utilizado para cada sub característica ou tipo de manutenção.

3.2 Qualidade de Código CSS

Existem poucos trabalhos que abordam a qualidade de código CSS na literatura e, como Mesbah e Mirshokraie (2012) identificaram, não existem trabalhos que analisem o código em função da sua manutenibilidade.

Existem trabalhos como o de Keller e Nussbaumer (2010), que analisam a qualidade de código CSS em uma perspectiva de avaliar a diferença em códigos de autoria humana e os gerados de forma automática, enquanto Mesbah e Mirshokraie (2012) propõem uma ferramenta para auxiliar na manutenção de código encontrando regras inefetivas e removendo-as do código.

Keller e Nussbaumer (2010) propõem uma medida de qualidade do código CSS baseando-se na abstração do seletor. Esse trabalho é baseado no argumento de que o objetivo do código CSS é a reutilização de suas regras. A abstração do seletor é então definida pela sua utilização no escopo geral do HTML, considerando que seletores com id são os menos abstratos possíveis. Este trabalho não conseguiu encontrar uma relação forte com a complexidade de código CSS e o nível de abstração, de forma que os autores a consideraram uma medida fraca, se utilizada de forma exclusiva, deixando em aberto a proposta de métricas que a corroborem, ou cooperem com ela na medida de qualidade do código CSS.

Quint e Vatton (2007) analisam os requisitos necessários para construir uma ferramenta de manipulação de estilos, baseando-se nas estruturas principais da linguagem CSS. Discutindo os métodos e técnicas que podem atender a esses requisitos, auxiliando os autores *web* de forma eficiente. Este trabalho discute a necessidade do estilo em documentos *web* de uma forma geral, mas opta por focar no CSS por ser mais utilizado e ter uma estrutura mais simples que o XSL, por exemplo, que também possui alguns estudos de mesma natureza.

Park e Saxena (2013) investigam os erros cometidos pelas pessoas ao codificar HTML e CSS. Aplicando um método de análise, foi possível dividir, a partir de gravações de vídeos, em seguimentos as dificuldades enfrentadas e os erros cometidos pelos participantes da pesquisa. Os resultados encontrados demonstraram que é possível utilizar o *framework skills-rules-knowledge* para analise de erros nos códigos, enquanto proveem uma compreensão da origem destes erros, e sugerem formas de se aprimorar ferramentas de desenvolvimento *web* para o suporte ao aprendizado de HTML e CSS.

4 Metodologia

O primeiro passo da execução do trabalho consistirá na consolidação do conceito de manutenibilidade em código CSS. Pretende-se alcançar este objetivo por meio de referências bibliográficas e de uma pesquisa do tipo *survey* com desenvolvedores profissionais, com níveis de experiência variados. Essa pesquisa pretende identificar as propriedades da linguagem e as situações mais comuns que dificultam, ou facilitam, a manutenção de código CSS.

Após executada a pesquisa, as métricas serão consolidadas, propondo valores individuais para as características da linguagem e definindo um índice de qualidade a partir de um agrupamento desses valores. Dessa forma, serão obtidas informações suficientes para construir uma ferramenta que calcule a métrica de um código CSS.

Utilizando de ferramentas automatizadas, serão identificadas as relações entre o índice proposto e a quantidade de falhas identificadas em sistemas *web* de código aberto. Utilizando bases de projetos de software de código aberto, iremos analisar e fazer a referência cruzada, entre a pontuação alcançada pelo código CSS e o número de problemas reportados relacionados ao projeto do mesmo.

4.1 Questionário

A pesquisa *survey* (questionário) é uma forma de obtenção de dados ou informações sobre características, ações ou opiniões de um determinado conjunto de pessoas, indicado como representante de uma população-alvo, por meio de um instrumento de pesquisa, normalmente um questionário (FREITAS; OLIVEIRA, 2000).

O interesse de uma pesquisa desse tipo é produzir descrições quantitativas de uma população. No caso deste trabalho, o objetivo é identificar, de forma quantitativa, as características identificadas pelos desenvolvedores como sendo as que classificam a qualidade do código. Para tanto, será executado um questionário exploratório, com o objetivo de identificar os conceitos do CSS que são centrais para a associação de qualidade do código.

4.2 Proposta das Métricas

As métricas são identificadores numéricos baseados em características da linguagem. No caso do CSS, estas características serão definidas a partir dos resultados obtidos no questionário.

4.3 Avaliação dos resultados

A partir da métrica proposta, será desenvolvida uma ferramenta automática de calculo da métrica. O programa irá ler o arquivo CSS, identificar as regras definidas e, a partir da definição proposta,calcular para cada arquivo. Se for necessário para o calculo da métrica, os arquivos HTML também serão considerados no cálculo da métrica.

Com as métricas calculadas, será testada a aderência da métrica a projetos *open source*, utilizando o número de problemas relacionados a CSS — como *bugs* identificados nas bases de código aberto pelos colaboradores — como parâmetro de triangulação. Dessa forma, será construída uma base de dados para as análises estatísticas, que confrontarão os resultados esperados deste trabalho.

5 Resultados Preliminares

Referências

BIEMAN, J.; OTT, L. Measuring functional cohesion. **IEEE Transactions on Software Engineering**, IEEE, v. 20, n. 8, p. 644–657, 1994. ISSN 00985589. Disponível em: <http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=310673>. Citado 3 vezes nas páginas 9, 10 e 12.

DHAMA, H. Quantitative models of cohesion and coupling in software. **J. Syst. Softw.**, Elsevier Science Inc., New York, NY, USA, v. 29, n. 1, p. 65–74, abr. 1995. ISSN 0164-1212. Disponível em: <http://dx.doi.org/10.1016/0164-1212(94)00128-A>. Citado 2 vezes nas páginas 10 e 12.

FANTASAI; ATIKNS, T. **CSS Cascading and Inheritance Level 4**. 2015. Disponível em: <http://www.w3.org/TR/2015/WD-css-cascade-4-20150421/>. Acesso em: 27 de maio de 2015. Citado na página 7.

FREITAS, H.; OLIVEIRA, M. **O Método de pesquisa Survey**. 2000. 105–112 p. Disponível em: <http://www.rausp.usp.br/download.asp?file=3503105.pdf>. Citado na página 14.

GENEVES, P.; LAYAIDA, N.; QUINT, V. On the analysis of cascading style sheets. In: **Proceedings of the 21st international conference on World Wide Web - WWW '12**. New York, New York, USA: ACM Press, 2012. p. 809. ISBN 9781450312295. Disponível em: <http://dl.acm.org/citation.cfm?id=2187836.2187946>. Citado 2 vezes nas páginas 2 e 5.

HICKSON, I.; BERJON, R.; FAULKNER, S.; LEITHEAD, T.; NAVARA, E. D.; O'CONNOR, E.; PFEIFFER, S. **HTML5: A vocabulary and associated APIs for HTML and XHTML**. 2014. Disponível em: <http://www.w3.org/TR/html/introduction.html#a-quick-introduction-to-html>. Acesso em: 24 de maio de 2015. Citado 2 vezes nas páginas 3 e 5.

IEEE. **IEEE Standard Glossary of Software Engineering Terminology**. 1990. 1 p. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=159342>. Citado 2 vezes nas páginas 2 e 12.

KELLER, M.; NUSSBAUMER, M. Css code quality: A metric for abstractness or why humans beat machines in css coding. In: **Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the**. [S.l.]: IEEE, 2010. p. 116–121. ISBN 978-1-4244-8539-0. Citado 2 vezes nas páginas 1 e 12.

MARDEN, P.; MUNSON, E. Today's style sheet standards: the great vision blinded. **Computer**, v. 32, n. 11, p. 123–125, 1999. ISSN 00189162. Disponível em: <http://www.researchgate.net/publication/2955166_Today's_style_sheet_standards_the_great_vision_blinded>. Citado na página 2.

MCCABE, T. J.; BUTLER, C. W. Design complexity measurement and testing. **Commun. ACM**, ACM, New York, NY, USA, v. 32, n. 12, p. 1415–1425, dez. 1989. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/76380.76382>. Citado 2 vezes nas páginas 11 e 12.

MESBAH, A.; MIRSHOKRAIE, S. Automated analysis of css rules to support style maintenance. In: **Proceedings of the 34th International Conference on Software Engineering**. Piscataway, NJ, USA: IEEE Press, 2012. (ICSE '12), p. 408–418. ISBN 978-1-4673-1067-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2337223.2337272>. Citado 3 vezes nas páginas 1, 2 e 12.

PARK, T.; SAXENA, A. Towards a Taxonomy of Errors in HTML and CSS. **Proceedings of the ...**, p. 75, 2013. Disponível em: <http://dl.acm.org/citation.cfm?doid=2493394.2493405$\delimiter"026E30F$nhttp://dl.acm.org/citation.cfm?id=2493405>. Citado na página 13.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. Porto Alegre: AMGH Editora Ltda., 2010. ISBN 978-85-63308-00-9. Citado 4 vezes nas páginas 8, 9, 11 e 12.

QUINT, V.; VATTON, I. Editing with style. In: **Proceedings of the 2007 ACM symposium on Document engineering - DocEng '07**. New York, New York, USA: ACM Press, 2007. p. 151. ISBN 9781595937766. Disponível em: <http://dl.acm.org/citation.cfm?id=1284420.1284460>. Citado 2 vezes nas páginas 2 e 13.

RIAZ, M.; MENDES, E.; TEMPERO, E. A systematic review of software maintainability prediction and metrics. In: **Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement**. Washington, DC, USA: IEEE Computer Society, 2009. (ESEM '09), p. 367–377. ISBN 978-1-4244-4842-5. Disponível em: <http://dx.doi.org/10.1109/ESEM.2009.5314233>. Citado na página 12.

W3C. **CSS**. 2015. Disponível em: <http://www.w3.org/Style/CSS>. Acesso em: 21 de maio de 2015. Citado na página 5.

W3C. **HTML**. 2015. Disponível em: <http://www.w3.org/html/>. Acesso em: 24 de maio de 2015. Citado na página 3.

WHITMIRE, S. A. **Object Oriented Design Measurement**. 1st. ed. New York, NY, USA: John Wiley & Sons, Inc., 1997. ISBN 0471134171. Citado 2 vezes nas páginas 8 e 12.

ZUSE, H. **Software Complexity: Measures and Methods**. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1991. ISBN 0-89925-640-6. Citado na página 12.

ZUSE, H. **A Framework of Software Measurement**. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1997. ISBN 3110155877. Citado na página 12.

ÇELIK, T.; ETEMAD, E. J.; GLAZMAN, D.; HICKSON, I.; LINSS, P.; WILLIAMS, J. **Selectors Level 3**. 2009. Disponível em: <http://www.w3.org/TR/2009/PR-css3-selectors-20091215/>. Acesso em: 24 de maio de 2015. Citado na página 6.