



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

UMA MÉTRICA DE QUALIDADE E 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

**UMA MÉTRICA DE QUALIDADE E MANUTENIBILIDADE DE
CÓDIGO CSS**

BELO HORIZONTE
JULHO DE 2015

*“There is no emotion, there is peace.
There is no ignorance, there is knowledge.
There is no passion, there is serenity.
There is no chaos, there is harmony.
There is no death, there is the Force.”*
— *The Jedi Code (Based on the meditations of
Odan-Urr)*

Resumo

Escrever código CSS não é uma tarefa trivial, visto que algumas características da linguagem constantemente causam inconsistências arquiteturais que podem resultar em efeitos colaterais. Devido a essas características, podem ser identificados uma série de fatores que dificultam a construção, manutenção e evolução do código CSS. Estas etapas são essenciais durante o tempo de vida do *software*, e isso não é diferente para aplicações *web*. Sendo assim é necessário que seja encontrada uma forma de mitigar os possíveis impactos da modificação, ou evolução, do código CSS de um projeto *web*. Uma forma de diminuir os impactos dessa etapa é criar uma métrica de qualidade que identifique o nível de manutenibilidade do código CSS. Essa necessidade ainda não foi suprida por nenhum trabalho acadêmico, portanto, é necessário ainda identificar quais são os aspectos que definem a qualidade do CSS. Para identificá-los, foi construído um questionário exploratório para fazer o levantamento das características do CSS que impactam na qualidade do código, visando uma forma de quantificar a manutenibilidade do código CSS. A partir da definição dos critérios de qualidade, foi construído um *script* para o cálculo da métrica proposta, de forma a gerar dados para ela e o número de defeitos gerados a partir do código calculado. Através destes resultados é possível notar um progresso em direção à qualidade de código CSS.

Palavras-chave: CSS. manutenibilidade. métrica. qualidade de código. Engenharia de Software. web.

Lista de Figuras

Figura 1 – Exemplo de arquivo HTML válido.	3
Figura 2 – Estrutura de uma <i>tag</i>	4
Figura 3 – Exemplo de arquivo HTML válido.	4
Figura 4 – Exemplo de estrutura da árvore do DOM.	5
Figura 5 – Exemplo de uma folha de estilo.	6
Figura 6 – Exemplo de questão aplicada no questionário.	14
Figura 7 – Resultado da questão 2 do questionário Apêndice A.	17
Figura 8 – Resultado da questão 3 do questionário Apêndice A.	18
Figura 9 – Resultado da questão 9 do questionário Apêndice A.	18
Figura 10 – Média de dificuldade por nível de proficiência em cada uma das questões de escala.	20
Figura 11 – Distribuição de tamanho dos seletores	21
Figura 12 – Exemplo de resultado da execução do <i>script</i> de cálculo da métrica.	22
Figura 13 – Gráfico de frequência de codificação do Jenkins.	24
Figura 14 – Comparação do resultado total da métrica em relação ao número de tarefas criadas.	26
Figura 15 – Comparação do resultado da métrica do <i>style.css</i> em relação ao número de tarefas criadas.	28
Figura 16 – Composição do valor da métrica por cada critério.	29

Lista de Quadros

Quadro 1 – Classificação das características do CSS e nível de proficiência	16
Quadro 2 – Respostas abertas da questão número 6 do questionário. (Apêndice A) . . .	19
Quadro 3 – Versões utilizadas para execução dos testes.	25

Lista de Abreviaturas e Siglas

CSS	<i>Cascading Style Sheets</i>
DOM	<i>Document Object Model</i>
HTML	<i>Hypertext Markup Language</i>
ISO	<i>International Organization for Standardization</i>
W3C	<i>World Wide Web Consortium</i>
WWW	<i>World Wide Web</i>
XSL	<i>Extensible Stylesheet Language</i>

Sumário

1 – Introdução	1
1.1 Justificativa	2
1.2 Objetivos	2
2 – Fundamentação Teórica	3
2.1 HTML	3
2.2 CSS	5
2.2.1 Seletores	6
2.2.2 Efeito Cascata	7
2.3 Qualidade de <i>Software</i>	8
3 – Trabalhos Relacionados	10
3.1 Qualidade de Código Clássica	10
3.2 Qualidade de Código CSS	10
4 – Metodologia	12
4.1 Questionário	12
4.2 Proposta das Métricas	12
4.3 Avaliação dos resultados	13
5 – Desenvolvimento	14
5.1 Construindo o questionário	14
5.2 Resultados do Questionário	15
5.2.1 Nível de proficiência	15
5.2.2 Visão Geral	16
5.2.3 Questões Exploratórias	17
5.2.4 Cálculo dos pesos	19
5.3 Criação da métrica	20
6 – Avaliação da Métrica	24
6.1 Metodologia de Avaliação	24
6.2 Dados para Teste	25
6.3 Resultados	26
6.4 Apreciação da métrica	27
7 – Conclusão	30
7.1 Contribuições	30

7.2 Trabalhos Futuros	31
Referências	32
 Apêndices	 34
APÊNDICE A – Questionário	35

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 (BERNERS-LEE; FISCHETTI, 2000).

Inicialmente proposta por Håkon Wium Lie e Bert Bos, a linguagem *Cascading Style Sheet* (CSS) propunha a separação em um documento de conteúdo — o arquivo HTML — e um documento com a definição da aparência — o documento CSS — para as páginas *web* (LIE, 2005). Sendo um dos três padrões fundamentais da W3C¹ 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 mais complexo e em maior quantidade, fazendo com que a sua manutenibilidade se tornasse mais onerosa quando de alterações corretivas ou evolutivas (MESBAH; MIRSHOKRAIE, 2012). Escrever código CSS não é uma tarefa trivial, visto que algumas características da linguagem, como herança e especificidade de seletores, constantemente causam inconsistências arquiteturais que podem resultar em efeitos colaterais (WALTON, 2015). Essas inconsistências arquiteturais podem prejudicar o que Keller e Nussbaumer (2010) definem como efetividade e eficiência de código CSS.

- **Efetividade do código:** As propriedades de estilo, definidas no documento CSS, são efetivas se aplicadas aos elementos de conteúdo da forma desejada pelo desenvolvedor.
- **Eficiência do código:** Existem várias formas de se aplicar estilos aos elementos de conteúdo. Portanto, os códigos de CSS que podem ter o mesmo resultado ainda podem diferir significativamente. A eficiência de um código CSS significa implementar a atribuição de propriedades de forma a minimizar o esforço de codificar, manter e eventualmente reutilizar o código.

O efeito colateral descreve o fenômeno onde um agente que foi desenvolvido para afetar somente um escopo bem limitado acaba afetando um escopo muito maior. Folhas de estilo CSS têm escopo global, e toda regra pode afetar partes desconexas do site, por isso efeitos colaterais são muito comuns. Uma vez que a folha de estilo, usualmente consiste em uma coleção de regras altamente acopladas, totalmente dependentes na presença, ordem e especificidade de outras regras, até mesmos a menor mudança pode afetar a efetividade do código (WALTON, 2015).

¹ World Wide Web Consortium - <http://www.w3.org/>

1.1 Justificativa

Devido às características da linguagem, pode-se identificar uma série de fatores que dificultam a construção, manutenção e evolução do código CSS.

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). Pretende-se identificar então, uma medida de manutenibilidade para códigos CSS.

A manutenção e modificação de um *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 e pesquisadas (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 explorado 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 esse nível, foi feita neste trabalho, uma proposta de métrica de qualidade de código CSS, focando a manutenção do código.

1.2 Objetivos

Esta pesquisa possui os seguintes objetivos:

- Identificar os aspectos da linguagem CSS que qualificam uma folha de estilo;
- Analisar os aspectos levantados e propor uma medida para a folha de estilo;
- Fazer uma análise e avaliar a relevância da métrica proposta a outros indicativos de manutenibilidade de código.

A partir desses objetivos, pretende-se propor uma métrica de manutenibilidade para códigos CSS, colaborando com a medição de folhas de estilo e auxiliando nos processos de produção para a plataforma *web*.

2 Fundamentação Teórica

Neste capítulo será explorado os conceitos de HTML, CSS e Qualidade de código, com o objetivo de tornar claro os termos utilizados no desenvolvimento deste trabalho.

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>

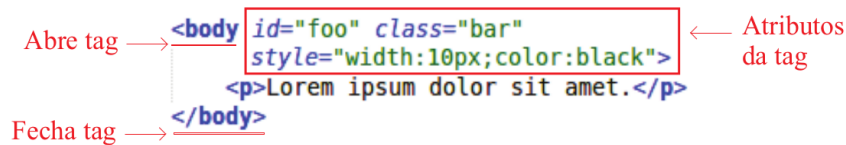
    <div id="div-bar" style="background-color:black; color:white;">
      <h2 class="header">Bar</h2>

      <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

Os atributos dos elementos têm o objetivo de organizar o arquivo HTML, ou definir seu estilo. Eles devem sempre ser definidos na *tag* de abertura, e são representados por um par chave/valor. Pode ser vista na Figura 2 uma estrutura simples de uma *tag* com os atributos *id*, *class* e *style* definidos.

Pode-se notar na Figura 1 a utilização dos atributos *id*, *title* e *style*, que represen-

Figura 2 – Estrutura de uma *tag*.

Fonte: Próprio autor

tam a identificação única do elemento, uma meta informação identificando a sua utilidade e o estilo — escrito na linguagem CSS — aplicado a ele, respectivamente.

Dentro do documento HTML pode-se utilizar as *tags* `<style/>` e `<script/>`, que definem escopos de código de estilo e linguagens de *script* de forma embarcada (*embedded*), 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

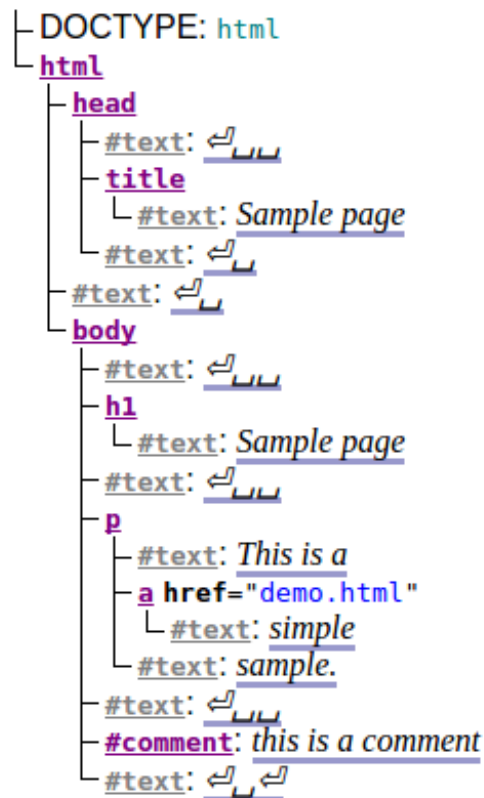
Além da definição *embedded*, pode-se definir estilos em CSS de outras duas formas, utilizando declarações *inline*, através do atributo `style` da *tag* HTML, ou referenciando a partir de um arquivo externo (*linked*), através do atributo `href` do elemento `<link>`.

É uma boa prática de desenvolvimento *web* manter as regras CSS em arquivos separados, evitando ao máximo a utilização das *tags* de definição de escopo, vistas na Figura 3. Essa separação mantém uma organização do código, possibilita o reuso em outras páginas, ou até mesmo em outros sistemas, e também melhora o desempenho, pois os arquivos podem ser mantidos em *cache*, diminuindo a carga de dados necessária para renderização de uma página *web*. Essa separação deve ser feita com cuidado, uma vez que a identificação das causas de possíveis erros se torna mais complexa.

Os navegadores *web* traduzem o arquivo HTML 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 (HICKSON et al., 2014).

Figura 4 – Exemplo de estrutura da árvore do DOM.



Fonte: Hickson et al. (2014)

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, em que todos os nós que se encontram nos níveis abaixo deste são denominados descendentes (*descendants*). Dentro da árvore DOM, os nós que se encontram exatamente um nível abaixo são os filhos (*childs*) e os nós que se encontram no mesmo nível são chamados de irmãos (*siblings*). Os nós denominados de `text` são os que encapsulam os textos inseridos dentro dos elementos HTML.

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 serão afetados.

2.2 CSS

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

Como pode ser visualizado na Figura 5, uma folha de estilo 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 a qual elas pertencem (GENEVES et al., 2012).

Figura 5 – Exemplo de uma folha de estilo.

O diagrama mostra um exemplo de uma folha de estilo CSS. No topo, uma regra completa R é destacada por um retângulo vermelho: `body p { color: blue; padding: 1px; background-color: gray; }`. Abaixo, três regras simples são mostradas com anotações:
 1. A regra `div > ul li { text-decoration: none; list-style: none; }` tem o seletor S_i apontando para `div > ul li`.
 2. A regra `#tabelaInicial > tr { font-weight: 100; font-family: Arial; }` tem a propriedade P_i apontando para `font-weight: 100` e o valor V_i apontando para `100`.
 3. A regra `.titulo { color: white; background-color: black; font-size: 25px; }` não possui anotações.

Fonte: Próprio autor

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 do HTML: o seletor universal, representado pelo símbolo `*`, 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 símbolo `#`. 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:

$$expression * element \rightarrow boolean \quad (1)$$

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, sendo este representado pelo sinal de maior (>), *e.g.* "body > p". Já o combinador de irmãos descreve os elementos que estão no mesmo nível da árvore, sendo eles representados em 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. Existem também os seletores de atributos, que selecionam elementos que possuam determinados atributos, permitindo a utilização de expressões para seleções parciais, *i.e.*, atributos cujos valores 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 aumenta. Além da complexidade dos seletores, pode-se apontar o efeito cascata, gerado pelo mecanismo de precedência e aplicação das propriedades aos elementos HTML, cujo funcionamento será descrito a seguir.

2.2.2 Efeito Cascata

A cascata em CSS se dá à atribuição de pesos a cada regra de estilo. Quando houverem várias regras candidatas a se aplicar, a de maior valor será escolhida. 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; ATKINS, 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 (!important).
- **Escopo:** Uma regra pode ter uma subárvore do DOM como escopo, afetando somente os elementos pertencentes a esta subárvore. Para regras normais, o escopo mais interno tem prioridade, para as regras definidas como importantes as do escopo mais externo sobrescreverão.
- **Especificidade:** O cálculo 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 é 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 possuem maior prioridade. Se duas propriedades possuem o valor `!important`, a ordem de precedência de renderização normal será aplicada.

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 `inherits`.

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 a 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.

3 Trabalhos Relacionados

Para discutir-se qualidade de código CSS, faz-se necessário o entendimento da qualidade de código fonte para os estudiosos de computação. A partir da perspectiva de trabalhos relevantes na área de conhecimento da qualidade de código fonte, será possível argumentar com uma base sólida as hipóteses e resultados encontrados nessa pesquisa.

Este capítulo introduzirá aspectos necessários para a discussão da qualidade de código fonte CSS.

3.1 Qualidade de Código Clássica

A norma ISO 9126 define seis atributos-chave de qualidade de *software* de computador, em que 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*, como por exemplo o trabalho de Whitmire (1997), que discute os atributos chave que definem a qualidade de um sistema de *software*, enquanto outros trabalhos exploram as medições de coesão, acoplamento e complexidade, que compõem os nove atributos-chave (MCCABE; BUTLER, 1989; ZUSE, 1991; BIEMAN; OTT, 1994; DHAMA, 1995; ZUSE, 1997) .

Riaz et al. (2009) discutem os trabalhos relacionados a métricas e previsão de manutenibilidade de código fonte. Por meio de uma revisão bibliográfica, eles 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.

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) analisam a qualidade de código CSS sob uma perspectiva de avaliar a diferença entre códigos de autoria humana e os gerados de forma automática. Propondo 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 de um documento HTML, considerando que seletores com `id` são os menos abstratos possíveis. O trabalho não conseguiu encontrar uma relação forte entre 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 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. Além disso, também discutem os métodos e técnicas que podem atender a esses requisitos, auxiliando os autores *web* de forma eficiente. O trabalho discute a necessidade das linguagens de 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 XSL¹, 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 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 análise 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.

Verificou-se, através dos trabalhos citados, a carência da medição de qualidade do código CSS. Como exposto por Park e Saxena (2013), o trabalho de codificar CSS não é trivial e necessita de suporte para o seu melhor desenvolvimento. Mas diferente do objetivo de Keller e Nussbaumer (2010), pretende-se com esse trabalho identificar as maiores dificuldades na manutenção de CSS.

Motivado pela escassez de trabalhos com esse objetivo, como explicitado por Mesbah e Mirshokraie (2012), elaborou-se uma métrica de qualidade para código CSS visando a sua manutenibilidade, um dos atributos-chave da ISO 9126. Utilizando o conceito de efeito colateral, exposto por Walton (2015), como principal argumento para identificação de códigos de difícil manutenção.

¹ Extensible Stylesheet Language

4 Metodologia

O primeiro passo da execução do trabalho consiste na conceituação de manutenibilidade em código CSS. Pretende-se alcançar este objetivo por meio de referências bibliográficas e de uma pesquisa do tipo *survey* aplicada a pessoas que escrevem código CSS em seu dia a dia, 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.

A partir dos resultados do questionário, pretende-se identificar os critérios de avaliação que impactam na qualidade do código CSS. Com a análise e os resultados obtidos para os critérios, de acordo com as hipóteses levantadas pelo autor, serão determinados os pesos de cada critério, para a execução dos testes individuais das folhas de estilo. Nesta etapa, será obtido um resultado numérico representando o cálculo dos pesos para os critérios encontrados no 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, será feita uma análise e a referência cruzada, entre o valor obtido pela métrica do código CSS e o número de problemas reportados relacionados ao projeto.

4.1 Questionário

A pesquisa *survey* 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 CSS. Para tanto, será aplicado 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, essas 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 cálculo da métrica. O programa irá ler o arquivo CSS, identificar as regras definidas e, a partir da definição proposta, calcular o valor obtido por cada arquivo. Se for necessário para o cálculo da métrica, os arquivos HTML também serão considerados.

Com as métricas calculadas, será testada a aderência da métrica a alguns projetos de código aberto, utilizando um dos indicadores de manutenibilidade disponível para o projeto, *e.g.*, número de defeitos, tempo gasto em correções, quantidade de modificações no código, etc. Dessa forma, será construída uma base de dados para as análises estatísticas, que confrontarão os resultados esperados deste trabalho.

5 Desenvolvimento

A partir de uma análise sobre qualidade de *software*, métricas de qualidade e fundamentos teóricos do funcionamento do CSS, construiu-se uma pesquisa exploratória, em forma de um questionário, para identificação dos aspectos mais relevantes no processo de manutenção de uma folha de estilo e das características do código fonte que estão relacionadas a sua qualidade.

5.1 Construindo o questionário

Elaborou-se o questionário com os seguintes objetivos:

- Identificar os aspectos da linguagem que mais impactam na legibilidade do código;
- Identificar os parâmetros que definem qualidade de código no ponto de vista dos entrevistados;
- Identificar aspectos mais custosos para manutenção;

A partir da coleta das respostas, pode-se analisar os pesos de cada aspecto de qualidade do código CSS em função de sua manutenibilidade. Identificando as maiores ocorrências de efeitos colaterais, quais são as técnicas para manutenção da legibilidade mais utilizadas e quais são os aspectos de organização do código mais relevantes.

Figura 6 – Exemplo de questão aplicada no questionário.

13.

```
div[id^="apply_form"] {
  margin-bottom:15px;
}
```

Muito Simples					Muito Complexo
1	2	3	4	5	
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	

Fonte: Próprio autor

Viu-se necessária a avaliação da complexidade de alguns aspectos da linguagem a partir do ponto de vista do profissional, para tal, o questionário (Apêndice A) foi construído com uma seção onde é avaliada, com base em um trecho de código (Figura 6), a dificuldade de se dar manutenção no mesmo. Cada trecho de código foi elaborado de acordo com um aspecto da linguagem que possa causar algum tipo de complicação. Esses aspectos foram escolhidos de acordo com as ponderações e experiência do autor, com base nos estudos realizados.

A dificuldade atribuída por cada pessoa a um determinado conjunto de regras, e propriedades, é subjetiva e depende fortemente da experiência do indivíduo. Portanto construiu-se o questionário com perguntas visando a classificação do respondente de acordo com o seu nível de conhecimento. A partir dessa classificação será possível ponderar as respostas de acordo com o nível de proficiência dos respondentes.

5.2 Resultados do Questionário

O questionário (Apêndice A) somou um total de vinte e sete (27) respostas. Este número de respostas pode ser atribuído ao alcance dos meios de divulgação, ou seja, não se fez uso de um canal de comunicação de uso da comunidade de desenvolvedores CSS. Mesmo com um pequeno número de respostas, pôde-se executar uma análise a partir dos resultados da pesquisa.

Durante os estudos para construção do questionário foram levantadas as seguintes hipóteses:

- **(h0)** A manutenção de folhas de estilo não é um trabalho trivial, podendo ocorrer efeitos colaterais durante esta etapa;
- **(h1)** O tamanho da folha de estilo é inversamente proporcional à manutenibilidade;
- **(h2)** Seletores com alta especificidade prejudicam a manutenção da folha de estilo;
- **(h3)** O uso correto de classes, com nomes coerentes, pode ser benéfico para a manutenção;
- **(h4)** A herança de propriedade é um fator causador de efeitos colaterais na etapa de manutenção;
- **(h5)** Seletores de alta complexidade prejudicam na manutenção;
- **(h6)** Regras que não são comumente utilizadas na construção de código CSS podem dificultar a manutenção.

O questionário teve então o intuito de validar essas hipóteses, de modo a confirmá-las ou refutá-las. Com uma série de perguntas exploratórias e as específicas, para determinar um valor de escala para os atributos específicos da linguagem.

5.2.1 Nível de proficiência

A primeira questão do questionário (Apêndice A) foi desenvolvida com o intuito de classificar os conhecimentos de cada respondente, para assim determinar o seu nível de proficiência. Essa classificação permitiu que os pesos definidos por cada respondente fosse ponderado no resultado final do questionário.

Quadro 1 – Classificação das características do CSS e nível de proficiência

Iniciante	Intermediário	Avançado
Localidade Agrupamento Aninhamento <i>Box Model</i>	Herança <i>Transformation</i> <i>Transition</i> Pseudo classes Pseudo elementos Especificidade	<i>At-rules</i> <i>Media queries</i> <i>Animation e keyframes</i>

Fonte: Próprio autor

Para esta classificação utilizou-se os critérios apresentados no Quadro 1, que indica as funcionalidades do CSS quanto ao seu nível de proficiência.

Cada critério recebeu um peso de acordo com sua categoria, sendo 1, 2 e 3 os valores para iniciante, intermediário e avançado, respectivamente. Um respondente foi considerado iniciante se somasse 12 ou menos em suas respostas, intermediário para o respondente que somasse de 13 a 20, e avançado caso suas respostas somassem 21 ou mais.

5.2.2 Visão Geral

As respostas para o questionário formaram um conjunto de dados capaz de validar as hipóteses levantadas, não de forma definitiva, mas com dados suficientes para construção da métrica.

As questões propostas para identificar os aspectos de qualidade do código CSS identificaram resultados diversos. Como na Figura 7, podemos verificar que mais de 90% dos respondentes identificaram os elementos estruturais, a nomenclatura coerente para as classes e identificadores (*id*) como sendo imprescindíveis para qualidade da folha de estilo.

Estas respostas corroboram com a hipótese h3, mostrando que a boa estruturação dos elementos do documento de conteúdo, impactam diretamente na construção da folha de estilo.

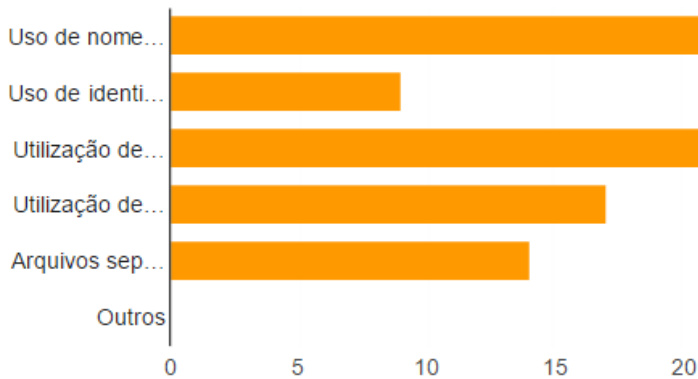
Na questão exposta na Figura 8 podemos notar os aspectos do CSS que interferem na qualidade do código. Nota-se aqui que não houve unanimidade para esta questão, porém percebe-se uma maior pontuação nas questões que têm impacto na legibilidade do código, *e.g.* organização em seções e modularidade do código.

Na Figura 9, é possível verificar que o maior número de ocorrências de efeitos colaterais, para os respondentes, está na fase de manutenção do código. Esse resultado corrobora com a hipótese (h0).

Ainda sobre os efeitos colaterais, foi elaborada uma questão com objetivo de identificar quais são as propriedades do CSS que mais os causam. As respostas foram variadas e por isso não foi possível identificar um padrão a partir desta pesquisa. No entanto, as respostas com elementos

Figura 7 – Resultado da questão 2 do questionário Apêndice A.

Quais dos aspectos a seguir são imprescindíveis no projeto de uma página HTML, para se criar a folha de estilo CSS com qualidade?



Uso de nomes significativos para classes (atributo class) e identificadores (atributo id)	25	96.2%
Uso de identificadores (atributo id) nos elementos	9	34.6%
Utilização de elementos estruturais (div's, span's etc.) para encapsulamento	24	92.3%
Utilização de classes	17	65.4%
Arquivos separados para organização das regras	14	53.8%
Outros	0	0%

Fonte: Próprio autor

semelhantes sempre tinham relação com definição de posicionamento e tamanho dos elementos (e.g.: `position`, `margin`, `padding`, `display`, `width`, `z-index`, `float`, etc). A partir das respostas obtidas pode-se identificar os valores de manutenibilidade para os elementos que possuem características de herança, prioridade e atuação semelhantes.

5.2.3 Questões Exploratórias

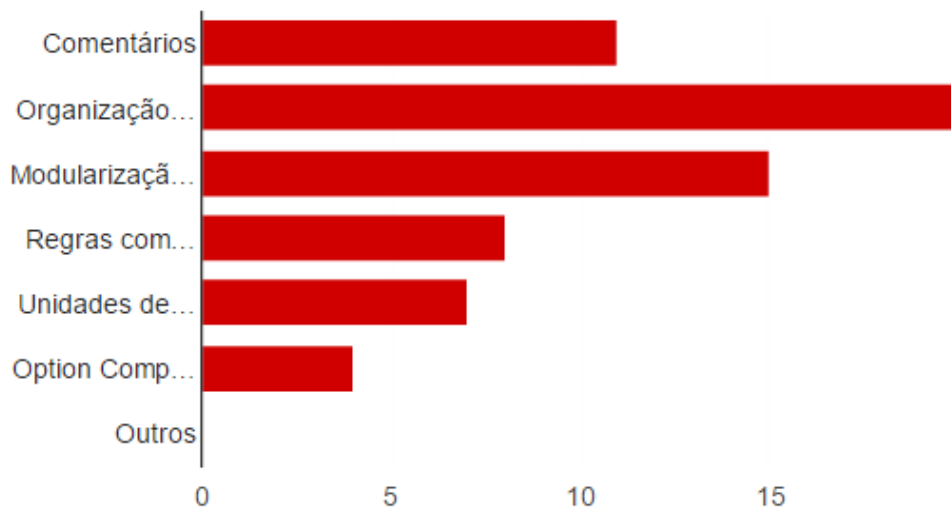
Algumas questões desse questionário tinham o objetivo de captar informações da experiência dos respondentes, a fim de identificar parâmetros que não foram cobertos pelo questionário. Algumas respostas agregaram valor à pesquisa, identificando esses pontos e mostrando algumas informações valiosas acerca da qualidade da folha de estilo.

Foi questionado quais são os pontos críticos que podem dificultar a manutenção, ou evolução, do código CSS. Das respostas obtidas, pode-se notar no Quadro 2 aquelas que corroboram ou refutam as hipóteses levantadas.

Essas respostas foram de suma importância para identificação de quais os critérios que deveriam, ou não, ser considerados para a avaliação da folha de estilo. Depois de identificados esses critérios, foi feita a análise das questões com exemplos de código, com o objetivo de definir

Figura 8 – Resultado da questão 3 do questionário Apêndice A.

Durante a construção de uma folha de estilo CSS, quais das opções interferem diretamente na qualidade do código?



Comentários	11	42.3%
Organização dos seletores em seções	20	76.9%
Modularização das regras	15	57.7%
Regras complexas	8	30.8%
Unidades de medida flexíveis (% , vw , vh , etc.)	7	26.9%
Option Compactação das regras (agrupamentos, utilização de atalhos)	4	15.4%
Outros	0	0%

Fonte: Próprio autor

Figura 9 – Resultado da questão 9 do questionário Apêndice A.

Efeitos colaterais ocorrem em maior quantidade em que fase da construção de uma página?



Fonte: Próprio autor

Quadro 2 – Respostas abertas da questão número 6 do questionário. (Apêndice A)

Resposta	Corrobora	Refuta
Estilizar elementos sem classe, criar folhas de estilos muito extensas.	h1	
CSS's que são atribuídos de forma mais genérica aos elementos.		h2
Regras complexas Herança de valor de propriedade (valores, inherit, initial) Aninhamento (seleção de elementos aninhados)	h5	
Utilização de nomes muitos genéricos para classes ou atributos. A estilização que não é mais usada e fica no código.	h3;h6	
Regras para itens muito genéricos. Utilização de !important. Código repetido.	h6	h2
Saber se um seletor está ou não sendo usado em algum parte do código.	h6	
Seletores muito específicos.	h2	
Regras com seletores muito gerais, como classes e tags, costumam provocar efeitos colaterais com mais frequência. Acho que para poder escrever regras desse tipo (gerais), todas as propriedades sendo definidas precisam ser "óbvias" (fácil de uma 3ª pessoa entender por que ela está ali) e também gerais (não sendo algo como uma classe .button definindo um left:54px, que deveria estar sendo aplicado a apenas um .button em particular e não a todos).	h2	
Arquivo desorganizado, regras repetidas, sem sessões definidas, código compactado.	h3;h6	

Fonte: Próprio autor, a partir de respostas do questionário

os seus pesos.

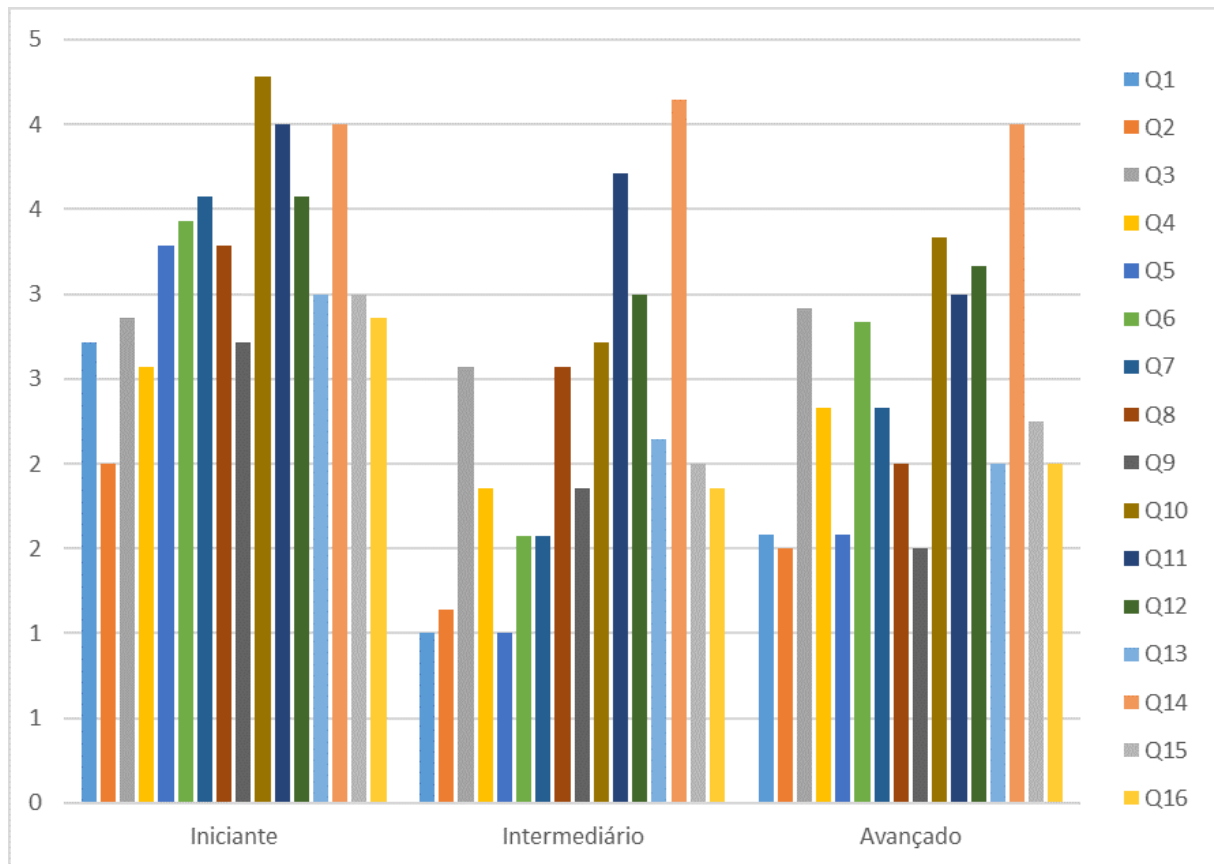
5.2.4 Cálculo dos pesos

A última seção de perguntas no questionário foi desenvolvida com a intenção de identificar as dificuldades dos respondentes, quando deparados com uma situação de código CSS. As propriedades e regras exemplificadas foram desenvolvidas com objetivos específicos, cada uma delas cobrindo uma das características que representavam, na visão do autor, um fator dificultador na modificação de uma folha de estilo.

Cada respondente identificou em uma escala (1 a 5) a dificuldade de se dar manutenção no trecho de código apresentado. Fez-se então uma média de cada resposta, para determinarmos a dificuldade de cada uma das 16 questões. Pode-se notar na Figura 10 a diferença de dificuldade encontrada pelos respondentes com diferentes níveis de proficiência.

É possível notar a proximidade das respostas dos níveis intermediário e avançado, mas ainda assim as médias do nível avançado são ligeiramente maiores que as do nível intermediário. Os pesos de cada critério foi identificado a partir da média ponderada de cada questão, para evitar

Figura 10 – Média de dificuldade por nível de proficiência em cada uma das questões de escala.



Fonte: Próprio autor

que o peso de cada resposta fosse completamente dependente das respostas dos respondentes identificados como iniciantes.

5.3 Criação da métrica

A partir das hipóteses levantadas e dos resultados obtidos no questionário, foram identificados pesos para os critérios de avaliação, como visto na Tabela 1. Esses pesos são utilizados para o cálculo de cada métrica de forma individual, aplicando uma forma de avaliação correspondente a cada um dos critérios.

Para o cálculo do critério de comprimento do seletor levou-se em consideração um valor de ativação, *i.e.*, a partir de dado número de caracteres o seletor terá um valor de manutenibilidade. Para este valor de ativação utilizou-se dos dados levantados pela pesquisa feita por McPherson (2014), em que são expostos alguns dados sobre a utilização do CSS na internet. Nesta pesquisa é apresentada uma distribuição do comprimento do seletor em caracteres, como visto na Figura 11. Esta distribuição mostra que a moda dos comprimentos encontrados pela pesquisa é em torno de vinte (20) caracteres. A decisão tomada para o valor de ativação do critério foi então definida como trinta e cinco (35) caracteres, um valor maior que a moda e que possui uma fatia expressiva

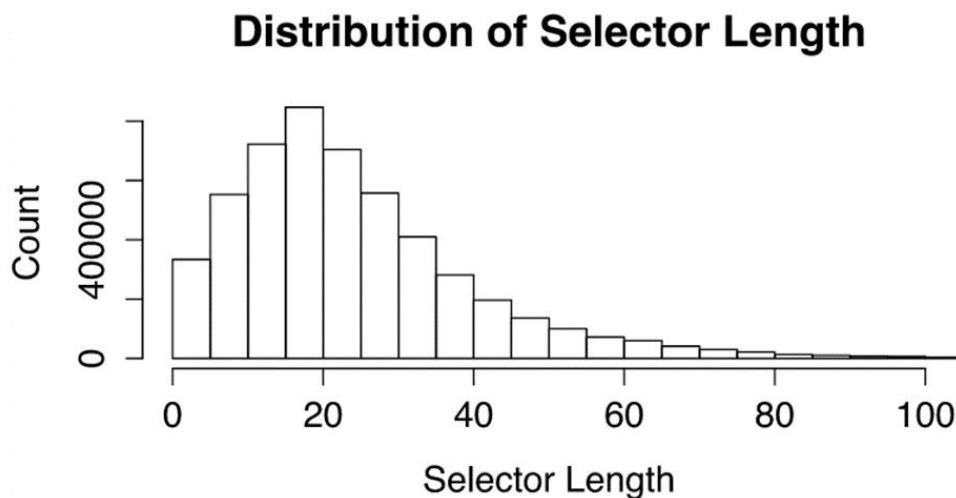
Tabela 1 – Tabela com peso de cada critério avaliado.

Critério	Peso
Seletores raros: {[^=], [\$=], , +,>}	3
Agrupamentos	2,8
Aninhamento	
Propriedade simplificada	3,2
Pseudo elementos	2,8
Seletor com mais de 20 caracteres	3
At-rules	2,8
Media queries	3,8
Prefixos: {-webkit, -ms, etc.}	4,2
Clausula :not	3,8
Complexidade do seletor	4,8
Seletor de localidade: {nth-last-child, first-child, etc.}	2,6

Fonte: próprio autor

na distribuição apresentada.

Figura 11 – Distribuição de tamanho dos seletores



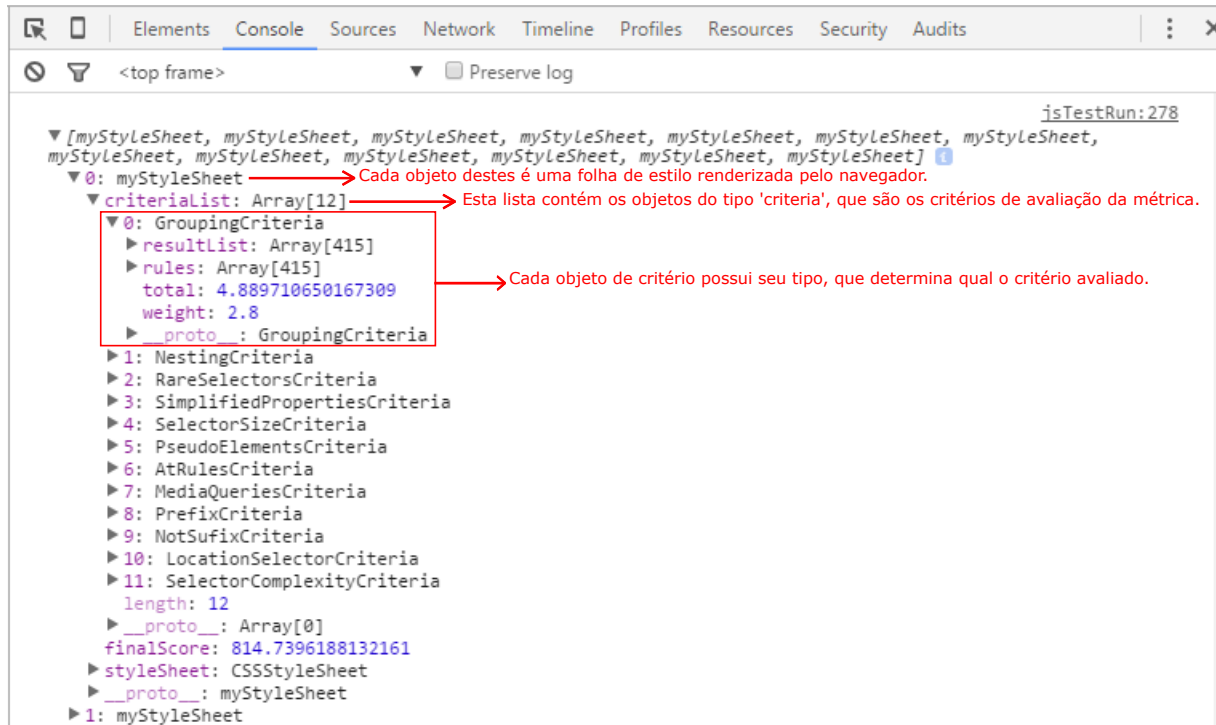
Fonte: McPherson (2014)

Outros dois critérios de avaliação com cálculos peculiares foram o agrupamento e a complexidade dos seletores. Para o critério de agrupamento foi considerado que a sua manutenibilidade tem um nível de saturação, para esta saturação considerou-se o agrupamento de vinte (20) seletores simples, definido arbitrariamente tomando por base o tamanho modal dos seletores apresentados pela pesquisa de McPherson (2014). A complexidade deste seletor foi calculada como um crescimento exponencial, onde o peso da dificuldade deste critério é potencializado pela posição do item agregador de complexidade.

Os demais critérios avaliados são calculados pelo número de repetições, faz-se a multiplicação do peso pela quantidade de ocorrências do critério e se obtém o resultado.

Para avaliação da métrica foi criado um *script* que lê a partir do DOM as regras aplicadas sobre o HTML renderizado no navegador. Uma vez que o *script* é executado no navegador, torna-se dependente deste a execução dos testes. Foram necessários alguns testes do *script* para ajustes dos pesos, esses testes foram executados em várias páginas da *web*.

Figura 12 – Exemplo de resultado da execução do *script* de cálculo da métrica.



Fonte: Próprio autor

O *script* executa os cálculos de cada um dos critérios para cada folha de estilo renderizada, iterando sobre as regras CSS da folha de estilo. Desta forma uma mesma regra será avaliada sobre todos os critérios determinados, uma vez que as regras podem se encaixar em mais de um critério. Ao final da execução do *script* será obtido o resultado total de cada critério em cada folha de estilo, o que permite uma análise da contribuição dos critérios para o resultado final da métrica.

Na Figura 12 pode-se visualizar a estrutura do resultado de uma execução do *script*. O código fonte desse *script* está disponível no repositório GitHub¹. O objetivo final deste código é gerar um *bookmarklet* que exiba as informações de qualidade do código CSS presente na página *web*.

Com o *script* para cálculo da métrica pronto e ajustado, fez-se a escolha de roteiro de testes. Para determinar qual seria a melhor forma de avaliar os resultados do *script* foi necessário escolher projetos em que o estilo da página fosse codificado em arquivos CSS, eliminando a possibilidade de aplicar os testes em projetos que utilizem preprocessadores²CSS. Além desta

¹ Disponível em <<https://github.com/vcsalvador/jsTestRun>>.

limitação, deveria ser possível analisar um outro indicador de manutenibilidade de código, como tempo de evolução, número de defeitos ou algum índice de retrabalho.

² Linguagens de programação intermediárias que geram código CSS, *e.g.* Sass, Less, Stylus, etc.

6 Avaliação da Métrica

O Jenkins¹, uma aplicação de integração contínua de código aberto, foi escolhido por atender a todas as limitações identificadas. O Jenkins é um projeto maduro, largamente utilizado e com uma comunidade de desenvolvimento ativa. Pode-se ver na Figura 13 a quantidade de adições e deleções no código por semana. Ele também foi escolhido como objeto de estudo por utilizar somente código CSS para a construção de suas folhas de estilo.

Figura 13 – Gráfico de frequência de codificação do Jenkins.



Fonte: GitHub, disponível em <<https://github.com/jenkinsci/jenkins/graphs/code-frequency>>. Acessado em 24/10/2015.

6.1 Metodologia de Avaliação

Para execução dos testes utilizamos doze (12) versões do Jenkins, selecionadas entre as que estão disponibilizadas para download na página do projeto, em intervalos semestrais, para obter dados históricos da aplicação. Essas versões foram escolhidas entre 2010 até a versão mais atual, esta escolha foi feita devido a disponibilidade e possibilidade de identificar em qual ponto no tempo elas foram construídas, conforme exibido no Quadro 3.

O projeto do Jenkins disponibiliza um meio de se cadastrar e controlar as tarefas relativas ao desenvolvimento da ferramenta (JIRA²). Utilizando dos filtros disponíveis no JIRA foi

¹ Disponível em <<https://jenkins-ci.org/>>

Quadro 3 – Versões utilizadas para execução dos testes.

Versão	Data de lançamento
1.369	31/07/2010
1.395	22/01/2011
1.423	25/07/2011
1.450	30/01/2012
1.475	01/08/2012
1.500	26/01/2013
1.525	29/07/2013
1.549	26/01/2014
1.574	27/07/2014
1.598	25/01/2015
1.622	27/07/2015
1.633	11/10/2015

Fonte: Próprio autor

possível identificar as tarefas que tinham alguma relação com os arquivos CSS, tornando possível uma coleta de dados indicadores do período de manutenção do projeto. Devido às características do JIRA, não foi possível a coleta do tempo gasto em cada tarefa, então o indicador escolhido foi a quantidade de defeitos criados a partir da versão de teste.

Com estas informações é possível identificar uma relação entre o valor da métrica e o número de defeitos de cada versão e, a partir disto, determinar o comportamento temporal do código CSS.

6.2 Dados para Teste

A partir do repositório³ de versões do Jenkins selecionou-se algumas de forma a cobrir uma janela de tempo razoável para as avaliações.

As versões foram executadas localmente e para cada versão o *script* de testes foi executado. Então o resultado de cada versão foi transcrito para uma planilha, cada uma delas contendo os resultados dos critérios para cada arquivo CSS encontrado pelo *script* e o número de defeitos encontrados pelo filtro do JIRA, entre a data de lançamento da versão de teste e a seguinte.

O resultado da métrica era então calculado como sendo o somatório dos resultados de todos os critérios de cada arquivo. O valor total da métrica encontrado para a página *web* é o somatório da métrica de todos os arquivos.

Para os testes na página principal da aplicação, identificou-se mais de um arquivo que compunham o estilo da página, para a métrica total foi utilizado todos os arquivos. Durante as

² Disponível em <<https://issues.jenkins-ci.org>>

³ Disponível em <<https://updates.jenkins-ci.org/download/war/>>

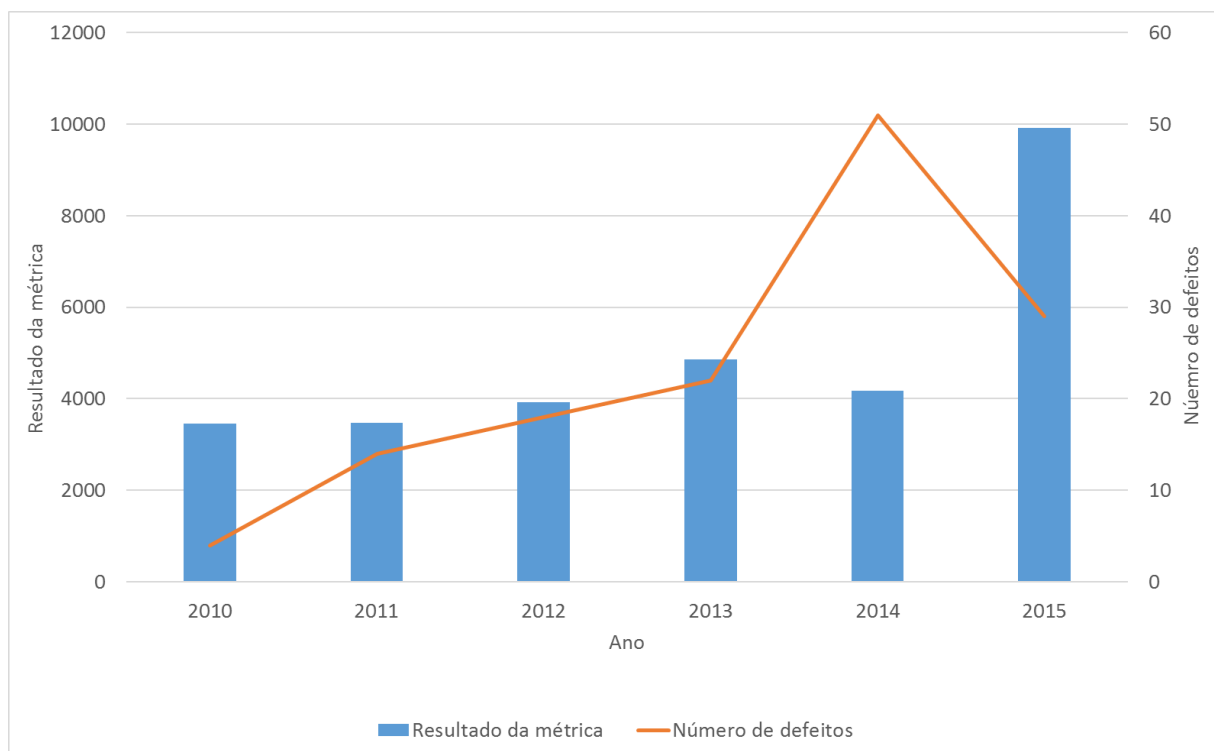
análises foram identificados arquivos de uma biblioteca de CSS, a YUI⁴. Os arquivos pertencentes à essa biblioteca influenciam os resultados das métricas e certamente na manutenibilidade do CSS da aplicação.

Os resultados da métrica para os arquivos da biblioteca YUI foram destoantes em relação aos vistos nos arquivos que realmente foram codificados pela equipe de desenvolvimento. Apesar desta grande diferença de pontuação, foi necessário analisar o resultado do agregado para construir uma análise da pontuação das folhas de estilo CSS carregadas na tela principal da aplicação, portanto incluindo os arquivos da biblioteca.

6.3 Resultados

Como as versões de teste foram escolhidas com intervalos semestrais, considerou-se a média delas como o valor da métrica para aquele ano. A partir desta média, foi feita uma análise da evolução do resultado e o número de defeitos abertos durante o ano.

Figura 14 – Comparação do resultado total da métrica em relação ao número de tarefas criadas.



Fonte: Próprio autor

Pode-se notar na Figura 14 que o número de defeitos acompanha a progressão da métrica, mas o pico de cadastro de defeitos não se encontra no mesmo intervalo em que há um pico do resultado da métrica. Este comportamento pode ser explicado pelo fato do *layout* da aplicação ter

⁴ YUI é uma biblioteca JavaScript e CSS, com objetivo de auxiliar na construção de aplicações *web* iterativas. Disponível em <<http://yuilibary.com/>>

sofrido uma mudança drástica entre as versões 1.549 e 1.574, disponibilizadas respectivamente em 26/01/2014 e 27/07/2014.

Devido ao valor elevado dos resultados vistos na Figura 14 considerou-se a possibilidade de isolar uma folha de estilo para análise do resultado, partindo do pressuposto que as modificações feitas pela equipe de desenvolvimento são as que impactam e definem a complexidade de manutenção do código. Para tanto foi necessária uma pesquisa no histórico de *commits* dos arquivos CSS do projeto, com intuito de identificar qual arquivo havia sofrido o maior número de alterações ao longo do tempo. Como pode ser visto na Tabela 2, o arquivo `style.css` foi o que sofreu mais *commits*, sendo identificado assim como o arquivo de codificação CSS principal do projeto.

Tabela 2 – Número de commits para cada arquivo CSS renderizado na página principal do Jenkins.

Arquivo CSS	Número de commits
<code>style.css</code>	113
<code>color.css</code>	2
<code>responsive-grid.css</code>	2
<code>yui\button.css</code>	3
<code>yui\container.css</code>	3
<code>yui\menu.css</code>	3

A partir desta identificação, a análise sobre progressão dos resultados e o número de defeitos criados foi refeita considerando somente o arquivo `style.css` com o intuito de determinar o impacto das modificações feitas na folha de estilo principal do projeto sobre o número de defeitos.

Pode-se notar na Figura 15 que a evolução da métrica foi crescente durante toda a progressão temporal e também um aumento considerável do resultado referente à mudança de *layout* em 2014.

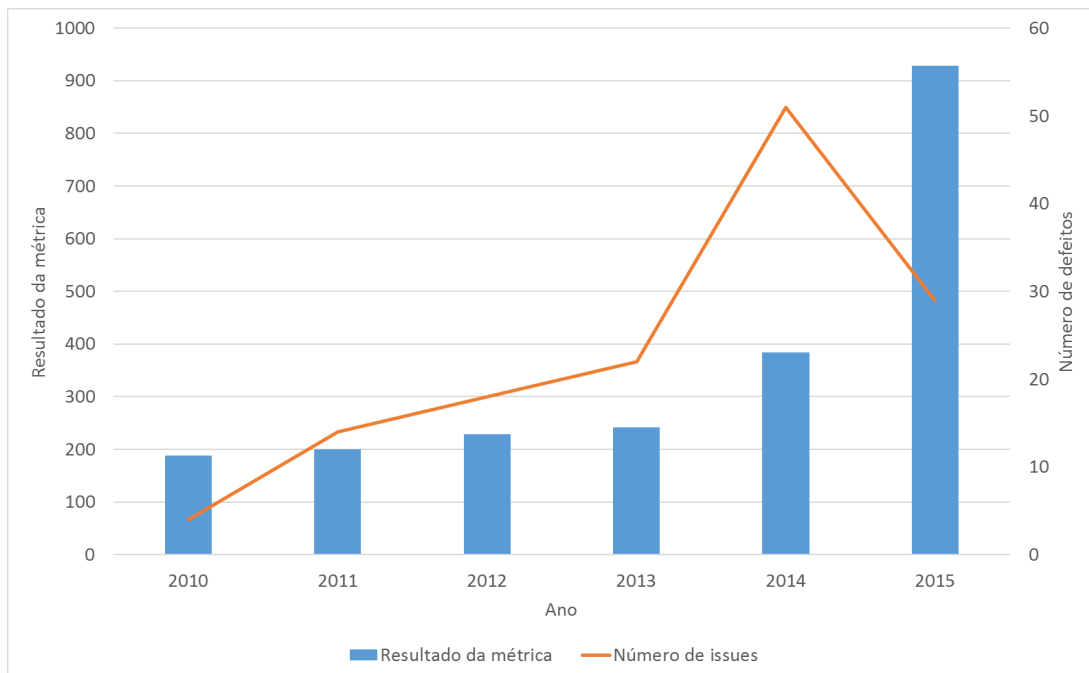
O comportamento do resultado da métrica para o arquivo `style.css` e o agregado é semelhante, o que leva a acreditar que o valor da métrica está relacionado às modificações feitas no projeto. Este comportamento para o arquivo `style.css` leva a hipótese de que as mudanças na folha de estilo causaram um aumento progressivo no resultado, ou seja, toda nova modificação somou ao valor da métrica.

Esta hipótese põe em questão qual o motivo desta mudança, que deve ser respondida através do estudo da métrica. Sobre quais são os critérios de avaliação que impactam no seu resultado e como esses se comportam.

6.4 Apreciação da métrica

Com o objetivo de avaliar a contribuição de cada critério no resultado final da métrica, fez-se uma análise da distribuição dentro do total de cada resultado. Na Figura 16 pode-se notar

Figura 15 – Comparação do resultado da métrica do `style.css` em relação ao número de tarefas criadas.



Fonte: Próprio autor

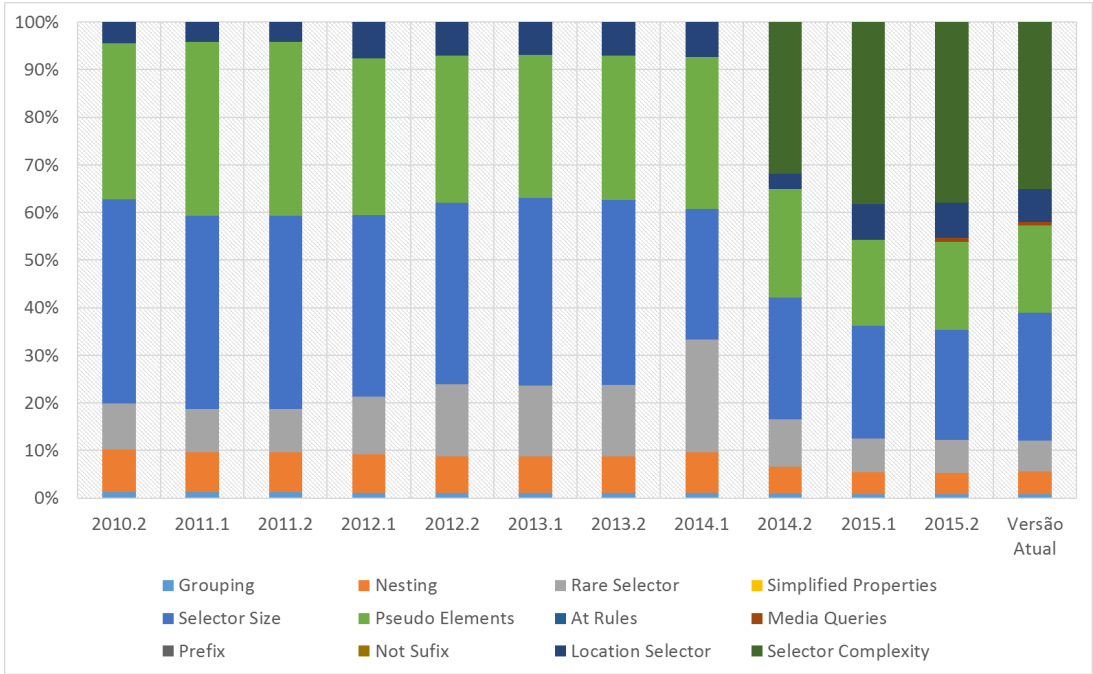
uma variação em função do tempo na parcela de contribuição dos critérios. Essa variação pode ser originada a partir da necessidade de se corrigir as regras, ou adicionar novas, para cada uma das situações encontradas durante o período de manutenção.

O aumento da contribuição de outros critérios no resultado da métrica, pode ser interpretado como um aumento na complexidade do código. Através da Figura 16, nota-se que o perfil de complexidade foi se alterando com a evolução do código, o que pode ser explicado pelo aumento da contribuição ou o surgimento de outros critérios que antes não estavam presentes na avaliação da métrica.

Esse aumento de complexidade do código CSS pode ser um agente causador de efeitos colaterais, que por sua vez podem causar um aumento considerável no número de defeitos no estilo de uma página *web*. Esse fato pode explicar o aumento gradativo dos números de defeitos criados acompanhando o resultado da métrica, visto na Figura 15.

Os resultados obtidos nos testes demonstram um aumento de complexidade do código ao longo do tempo, sendo que essa complexidade impacta no número de defeitos encontrados. Porém, os resultados obtidos não são determinantes, por falta de uma base de comparação. Pode-se concluir então que a métrica apresentada é um passo importante em direção à definição de qualidade de código CSS e pode ser usada como base de comparação para investigações mais profundas de outras aplicações.

Figura 16 – Composição do valor da métrica por cada critério.



Fonte: Próprio autor

7 Conclusão

A qualidade de código CSS é uma área de conhecimento inexplorada. A partir dos resultados encontrados nessa pesquisa é possível identificar um caminho à ser traçado para encontrar uma métrica definitiva.

O questionário permitiu a identificação das características que compõem a qualidade de código CSS e as dificuldades encontradas durante a etapa de manutenção deste. Pode-se notar que a legibilidade e os efeitos colaterais são aspectos que exercem influência para a qualidade geral do código CSS.

Os doze (12) critérios de avaliação identificados, a partir do questionário, permitiram calcular o nível de manutenibilidade do código CSS da aplicação Jenkins ao longo do tempo. A comparação dos resultados obtidos com o número de defeitos cadastrados no JIRA para cada versão do Jenkins testada, mostrou que o aumento do valor da métrica está relacionado com o aumento dos defeitos. Este fato demonstra que os parâmetros avaliados pelos critérios desenvolvidos está relacionada com a qualidade geral do código.

A partir de uma análise do histórico de *commits* dos arquivos CSS utilizados pela aplicação, permitiu a identificação do comportamento da qualidade em função das modificações feitas pela equipe no código. Foi então identificado que o arquivo CSS que mais sofreu alterações ao longo do tempo exibiu o mesmo perfil de evolução da métrica, o que leva a entender que a participação da equipe de desenvolvimento teve impacto direto na qualidade do código CSS.

Após as análises de impacto na qualidade dos resultados da métrica, identificou-se o perfil de composição dos critérios para o resultado final. Estas análises mostraram que as modificações ao longo do tempo impactaram no perfil de complexidade do código, indicados pelo surgimento de novos critérios de avaliação, ou aumento da participação total de outros.

O número de defeitos criados em uma aplicação não é a melhor forma de se determinar a manutenibilidade do código, mas pode ser um indicador do seu nível de qualidade. Portanto, os resultados encontrados não determinam uma métrica de manutenibilidade definitiva, mas iluminam o caminho para estudos futuros.

7.1 Contribuições

Esse trabalho modifica o estado de inércia de uma área de estudo que não é muito abordada na comunidade científica, a qualidade de código CSS em função de sua manutenibilidade. Partindo do zero conseguiu-se determinar doze (12) critérios de avaliação para a métrica de qualidade, que foram capazes de demonstrar um comportamento factível da quantidade de defeitos encontrados no *layout* de uma página *web*.

Durante a execução dos testes para identificação da métrica, construiu-se uma ferramenta de avaliação das folhas de estilo renderizadas em uma página *web*. O *script* construído é capaz de executar os testes dos critérios codificados e está disponível em um repositório aberto, para participação da comunidade e para utilização em trabalhos futuros.

Os resultados apresentados não formam uma métrica definitiva, mas representam um passo em direção à definição de qualidade de código CSS. Cabendo ainda a análise dos resultados sob a perspectiva de tempo em correções, ajustes nos pesos dos critérios e identificação de novos para avaliação. Sendo assim uma adição importante na área de Engenharia de Software, principalmente no que diz respeito à interface *web*.

7.2 Trabalhos Futuros

Este trabalho ilumina o caminho para futuras pesquisas sobre qualidade de código CSS. Então, com o objetivo de dar continuidade ao trabalho, alguns pontos foram levantados como possíveis trabalhos futuros.

Devido à popularização do uso de pré-processadores CSS, vê-se como necessário a identificação de uma avaliação da métrica para este tipo de código.

Para melhor identificar os indicadores de qualidade e manutenibilidade do código, deve-se acompanhar todo o processo de desenvolvimento de uma aplicação *web*, medindo o tempo gasto com a manutenção, evolução e correção do código CSS. Desta forma será possível identificar melhor o comportamento da métrica proposta com a etapa de manutenção do código.

Modificações no arcabouço desenvolvido para execução dos testes durante a fase de construção da aplicação, por exemplo, durante o processo de integração contínua.

Referências

BERNERS-LEE, T.; FISCHETTI, M. **Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor**. [S.l.]: HarperInformation, 2000. ISBN 006251587X. Citado na página 1.

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 na página 10.

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](http://dx.doi.org/10.1016/0164-1212(94)00128-A)>. Citado na página 10.

FANTASAI; ATKINS, 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 12.

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 6.

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 10.

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 3 vezes nas páginas 1, 10 e 11.

LIE, H. W. **Cascading Style Sheets**. Tese (Doutorado) — University of Oslo, mar. 2005. Citado na página 1.

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 na página 10.

MCPHERSON, A. **Quick Left Reports on Internet Performance**. 2014. Disponível em: <<http://reports.quickleft.com/css>>. Acesso em: 21 de agosto de 2015. Citado 2 vezes nas páginas 20 e 21.

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 4 vezes nas páginas 1, 2, 10 e 11.

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%delimater%026E30F%http://dl.acm.org/citation.cfm?id=2493405>>. Citado na página 11.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. Porto Alegre: AMGH Editora Ltda., 2010. ISBN 978-85-63308-00-9. Citado 3 vezes nas páginas 8, 9 e 10.

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 11.

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 10.

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.

WALTON, P. **Side Effects in CSS**. 2015. Disponível em: <<http://philipwalton.com/articles/side-effects-in-css/>>. Acesso em: 26 de setembro de 2015. Citado 2 vezes nas páginas 1 e 11.

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 10.

ZUSE, H. **Software Complexity: Measures and Methods**. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1991. ISBN 0-89925-640-6. Citado na página 10.

ZUSE, H. **A Framework of Software Measurement**. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1997. ISBN 3110155877. Citado na página 10.

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.

Apêndices

APÊNDICE A – Questionário

Olá,

Meu nome é Victor Salvador e estou desenvolvendo um trabalho de pesquisa com a finalidade de determinar uma métrica de qualidade de código fonte CSS para classificar sua manutenibilidade. Manutenibilidade de um código fonte é definida pela facilidade de modificá-lo, corrigi-lo ou evoluí-lo.

Este questionário tem por objetivo identificar o que, para os profissionais que escrevem código CSS, determina um código fonte de qualidade, de fácil leitura, que evite efeitos colaterais indesejados e que facilite futura manutenção corretiva ou evolutiva. Esta é uma pesquisa exploratória de opinião, portanto, não há respostas certas ou erradas.

* 1. Quais os aspectos de CSS que você conhece?

Marque apenas os itens que você consegue ler e compreender quando se depara com eles em um código.

- ☐ Herança de valor de propriedade (valores inherit, initial)
- ☐ Especificidade de seletores (precedência de seletores)
- ☐ Localidade (local onde a propriedade é definida: inline, <style>, .css)
- ☐ Box Model (funcionamento de elementos inline, block, inline-block)
- ☐ Pseudo classes (seletor de estado, :hover, :visited)
- ☐ Pseudo elemento (seleciona locais específicos do seletor, ::after, ::before)
- ☐ Agrupamento (aplicação da regra para um conjunto de seletores)
- ☐ Aninhamento (seleção de elementos aninhados)
- ☐ At-rules (@import, @media, @font-face)
- ☐ Media queries (e.g., @media screen and (max-width: 100px))
- ☐ Animações de valores de propriedades (animation e keyframes)
- ☐ Propriedade de transformação (transform)
- ☐ Transições entre valores de propriedades (transition)
- ☐ Outro (especifique)

* 2. Quais dos aspectos a seguir são imprescindíveis no projeto de uma página HTML, para se criar a folha de estilo CSS com qualidade?

- ☐ Uso de nomes significativos para classes (atributo *class*) e identificadores (atributo *id*)
- ☐ Uso de identificadores (atributo *id*) nos elementos
- ☐ Utilização de elementos estruturais (div's, span's etc.) para encapsulamento
- ☐ Utilização de classes
- ☐ Arquivos separados para organização das regras
- ☐ Outro (especifique)

3. Durante a construção de uma folha de estilo CSS, quais das opções interferem diretamente na qualidade do código?

- ☐ Comentários
- ☐ Organização dos seletores em seções
- ☐ Modularização das regras
- ☐ Regras complexas
- ☐ Unidades de medida flexíveis (% , vw, vh, etc.)
- ☐ Compactação das regras (agrupamentos, utilização de atalhos)
- ☐ Outro (especifique)

4. Como você prefere ver a ordem das regras CSS definidas em um arquivo?

- ☐ De acordo com o surgimento dos elementos a que se aplicam as regras no documento HTML
- ☐ De acordo com a ordem de criação das regras (novas regras sempre criadas ao final)
- ☐ De regras mais genéricas (tags, classes) no início para regras mais específicas (ids) no final do arquivo
- ☐ Outro (especifique)

5. Você usa alguma ferramenta preprocessadora de CSS (Sass, Less, Stylus, etc)

- ☐ Sim
- ☐ Não

* 6. Na manutenção de código CSS, quais são os pontos críticos que podem dificultar, ou são perigosos de se alterar?

* 7. Na manutenção de código CSS, quais são os pontos críticos que podem dificultar, quando você não participou da autoria?

* 8. Em um código CSS, propriedades podem ser herdadas, sobrescritas ou mesmo nunca aplicadas a um elemento HTML. Alterar uma propriedade CSS pode ter efeitos colaterais inesperados em outras partes do website/sistema. Isso acontece com alguma frequência com você?

* 9. Efeitos colaterais ocorrem em maior quantidade em que fase da construção de uma página?

- ☐ Criação
- ☐ Manutenção

10. Que tipos de características de um código CSS dificultam que você consiga fazer uma alteração em um código existente sem que aconteçam efeitos colaterais indesejados?

Na seção abaixo, identifique a complexidade do código CSS apresentado. Considere que um trecho de código é "muito simples" se você fizesse uma alteração nele sem nenhuma hesitação e "muito complexo" se você precisasse fazer bastante análise para alterá-lo.

11.

```
.element {  
  width:100px;  
  padding:10px;  
  border:1px solid #fff;  
}
```

Muito Simples

1

2

3

4

Muito Complexo

5

☐☐☐☐☐

12.

```
#justify {  
  text-align: justify;  
  width:100%;  
}
```

Muito Simples

1

2

3

4

Muito Complexo

5

☐☐☐☐☐

13.

```
div[id^="apply_form"] {  
  margin-bottom:15px;  
}
```

Muito Simples

1

2

3

4

Muito Complexo






5

☐☐☐☐☐

```

14. .icon-arrow-right, .icon-book, .icon-close, .icon-menu, .icon-mobile, .icon-rocket, .icon-signup, .icon-spin, .icon-twitter, .icon-user {
    font-family: jso-ico-font-0-3;
    font-style: normal;
    font-weight: 400;
    font-variant: normal;
}






```

Muito Simples 1	2	3	4	Muito Complexo 5
				

```

15. a:hover {
    text-decoration: none;
    color: blue;
    background-color: yellow;
}






```

Muito Simples 1	2	3	4	Muito Complexo 5
				

```

16. p {
    font: 14px/1.5 "Times New Roman", times, serif;
    padding: 30px 10px;
    border: 1px black solid;
    border-width: 1px 5px 5px 1px;
    border-color: red green blue yellow;
    margin: 10px 50px;
}






```

Muito Simples 1	2	3	4	Muito Complexo 5
				

```

17. li:before {
    background: red;
    color: #fc0;
}

```

Muito Simples 1	2	3	4	Muito Complexo 5
				

18.

```
input[type=email]:focus, input[type=number]:focus, input[type=password]:focus, input[type=search]:focus,
input[type=tel]:focus, input[type=text]:focus, input[type=url]:focus, textarea:focus {
  border-color: #8A8;
}
```

Muito Simples

1

2

3

4

Muito Complexo

5



19.

```
@font-face {
  font-family: "font of all knowledge";
  src: url(fontofallknowledge.woff);
}
```

Muito Simples

1

2

3

4

Muito Complexo

5



20.

```
@media(max-width:700px) {
  .intro .mega {
    font-size: 2.3125em;
    margin-bottom: .15em;
  }
  .intro .h2 {
    font-size: 1.3125em;
  }
}
@media(max-width:400px) {
  .intro .mega {
    font-size: 1.75em;
    margin-bottom: .25em;
  }
  .h2, .intro .h2, h2 {
    font-size: 1.3125em;
  }
}
```

Muito Simples

1

2

3

4

Muito Complexo

5



21.

```
input[type=number]::-webkit-inner-spin-button, input[type=number]::-webkit-outer-spin-button {
  -webkit-appearance: none;
  margin: 0;
}
```

Muito Simples					Muito Complexo
1	2	3	4	5	
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	

22.

```
#nav li a:not(#mobile-close-nav-btn) {
  font-size: 1.125px;
  font-weight: 700;
  padding: .25px .5px;
}
```

Muito Simples					Muito Complexo
1	2	3	4	5	
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	

23. .post-categories > li {

```
float: left;
margin-right: .5px;
```

}

Muito Simples					Muito Complexo
1	2	3	4	5	
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	

24.

```
div:nth-of-type(3) ~ ul:last-child li:nth-of-type(odd) *{
  font-weight:bold;
}
```

Muito Simples					Muito Complexo
1	2	3	4	5	
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	

25. <div class="quadro">

<div class="box-blue">

<p>

Lorem ipsum dolor sit amet, consectetur adipisicing elit, tempor incididunt ut labore et dolore magna quis nostrud exercitation ullamco laboris nisi ut aliquip ex consequat. Duis aute irure dolor in reprehenderit in cillum dolore eu fugiat nulla pariatur. <q>Excepteur sint proident, sunt in culpa qui officia deserunt </q>

</p>

```

</div>
<div class="box-yellow">
  <p>
    Lorem ipsum dolor sit amet, consectetur adipisicing elit,
    tempor incididunt ut labore et <b>dolore</b> magna
    quis nostrud exercitation ullamco laboris nisi ut aliquip ex
    consequat. Duis aute irure dolor in reprehenderit in
    cillum dolore eu fugiat nulla pariatur. <q>Excepteur sint
    proident, sunt in culpa qui <b>officia</b> deserunt </q>
  </p>
</div>
</div>

```

```

<style type="text/css">
.quadro {
  color: white;
  background-color: black;
}

.quadro.box-blue {
  color: red;
  background-color: blue;
}

.quadro.box-yellow {
  color: green;
  background-color: yellow;
}

.quadro q b {
  color: white;
}

.quadro p b {
  color: black;
}

.quadro q {
  color: gray;
}
</style>

```

Muito Simples

1

2

3

4

Muito Complexo

5



26.

```
.row-fluid:first-child {  
  padding: 10px;  
}
```

Muito Simples

1

2

3

4

Muito Complexo

5

