



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

EM DIREÇÃO A 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
NOVEMBRO DE 2015

VICTOR CARNEIRO SALVADOR

**EM DIREÇÃO A UMA MÉTRICA DE QUALIDADE E
MANUTENIBILIDADE DE CÓDIGO CSS**

BELO HORIZONTE
NOVEMBRO DE 2015

Centro Federal de Educação Tecnológica de Minas Gerais

Curso de Engenharia de Computação

Avaliação do Trabalho de Conclusão de Curso

Aluno: Victor Carneiro Salvador

Título do Trabalho: Em Direção a uma Métrica de Qualidade e Manutenibilidade de Código CSS

Data da defesa: 27/11/2015

Horário: 14:00

Local da defesa: Sala 101, Prédio 17 do CEFET-MG - Campus II

O presente Trabalho de Conclusão de Curso foi avaliado pela seguinte banca:

Professor Flávio Roberto dos Santos Coutinho - Orientador
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Professora Glívia Angélica Rodrigues Barbosa - Membro da banca de avaliação
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Professor Ismael Santana Silva - Membro da banca de avaliação
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Resumo

A qualidade de código CSS é uma área de conhecimento inexplorada. A codificação de folhas de estilo CSS não é uma tarefa trivial e para se chegar ao resultado desejado é necessário um certo esforço de tempo. Apesar de aparentar simplicidade, a linguagem CSS contém algumas armadilhas para os desenvolvedores, como o efeito cascata, herança de propriedades e especificidade. Devido a essa complexidade na codificação de CSS, a manutenção desse código se torna onerosa. Este trabalho identificou critérios de avaliação e pontos de interesse, em direção a uma métrica de qualidade, que viabilizam o estudo sobre a etapa de manutenção de códigos CSS. A partir de uma pesquisa exploratória foram identificados alguns aspectos da linguagem CSS que compõem a qualidade de código, do ponto de vista dos desenvolvedores, e a partir desses aspectos foi proposta uma métrica de qualidade para medir a manutenibilidade do código CSS. Para elaboração da métrica, foram definidos 12 critérios de avaliação, que representam algumas características da linguagem que apresentam dificuldade no momento de correção. Foi construído um calculador automático, desenvolvido em JavaScript, para avaliar o valor da métrica para qualquer página *web*. Como objeto de estudo foi escolhido o Jenkins, um sistema *web* de integração contínua, por utilizar somente arquivos CSS para codificação de seu estilo e possuir um gerenciador de tarefas público para identificação dos defeitos levantados ao longo de toda sua vida. A partir dos testes de versões do Jenkins, desde 2010 até a versão mais atual, foi calculada a métrica do seu código CSS ao longo do tempo, para comparar o comportamento da métrica em relação ao número de defeitos criados. Para verificar o comportamento da métrica em função das alterações feitas no código, foi escolhido o arquivo que sofreu maior número de *commits*. Foi possível notar que a métrica foi crescente ao longo do tempo, o que indica que a longevidade de um código CSS o torna mais complexo de se manter. O comportamento da métrica em relação ao número de defeitos é proporcional ao aumento da complexidade de manutenção. Trabalhos futuros contemplam consolidar a métrica, sendo necessárias mais iterações de testes e ajustes.

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 utilização das <i>tags</i> <code><style></code> e <code><script></code>	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.	15
Figura 7 – Resultado da questão 2 do questionário Apêndice A.	18
Figura 8 – Resultado da questão 3 do questionário Apêndice A.	19
Figura 9 – Resultado da questão 9 do questionário Apêndice A.	19
Figura 10 – Média de dificuldade por nível de proficiência em cada uma das questões de escala.	21
Figura 11 – Exemplo de propriedades simplificadas.	25
Figura 12 – Distribuição de tamanho dos seletores	26
Figura 13 – Exemplo de resultado da execução do <i>script</i> de cálculo da métrica.	28
Figura 14 – Gráfico de frequência de codificação do Jenkins.	29
Figura 15 – Comparação do resultado total da métrica em relação ao número de defeitos criados.	32
Figura 16 – Comparação do resultado da métrica do <code>style.css</code> em relação ao número de defeitos criados.	33
Figura 17 – Composição do valor da métrica por cada critério.	34

Lista de Tabelas

Tabela 1	– Tabela com peso de cada critério avaliado.	22
Tabela 2	– Tabela com os resultados do <i>script</i> de cálculo automático para a versão 1.369	31
Tabela 3	– Número de <i>commits</i> para cada arquivo CSS renderizado na página principal do Jenkins.	33

Lista de Quadros

Quadro 1 – Classificação das características do CSS e nível de proficiência	17
Quadro 2 – Respostas abertas da questão número 6 do questionário. (Apêndice A) . . .	20
Quadro 3 – Quadro com as funcionalidades exploradas em cada questão do questionário no Apêndice A	23
Quadro 4 – Tabela com versões utilizadas e o número de defeitos gerados em cada uma delas.	30

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	6
2.2.1 Seletores	6
2.2.2 Efeito Cascata	7
2.3 Qualidade de <i>Software</i>	8
3 – Trabalhos Relacionados	11
3.1 Qualidade de Código Clássica	11
3.2 Qualidade de Código CSS	11
4 – Metodologia	13
4.1 Questionário	13
4.2 Proposta da Métrica	13
4.3 Avaliação dos Resultados	14
5 – Desenvolvimento	15
5.1 Construindo o Questionário	15
5.2 Resultados do Questionário	16
5.2.1 Nível de Proficiência	17
5.2.2 Visão Geral	17
5.2.3 Questões Exploratórias	18
5.2.4 Cálculo dos Pesos	20
5.3 Criação da Métrica	21
5.3.1 Identificação dos Critérios de Avaliação	22
5.3.1.1 Seletores Raros	23
5.3.1.2 Agrupamento de elementos	24
5.3.1.3 Seletores Aninhados	24
5.3.1.4 Propriedades Simplificadas	24
5.3.1.5 Pseudo Elementos	24
5.3.1.6 Comprimento de Seletores	25
5.3.1.7 <i>At-rules</i>	25
5.3.1.8 <i>Media Queries</i>	25

5.3.1.9	Prefixos	26
5.3.1.10	Cláusula :not	26
5.3.1.11	Complexidade do Seletor	27
5.3.2	<i>Script</i> de Cálculo Automático da Métrica	27
6	– Avaliação da Métrica	29
6.1	Metodologia de Avaliação	29
6.2	Dados para Teste	31
6.3	Resultados	32
6.4	Apreciação da Métrica	34
7	– Conclusão	36
7.1	Contribuições	37
7.2	Trabalhos Futuros	37
	Referências	39
	 Apêndices	 41
	APÊNDICE A – Questionário	42

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 e até mesmo de sistemas de informação mais complexos, 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 das páginas *web* em um documento de conteúdo — o arquivo HTML — e um documento com a definição da aparência — o documento CSS (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 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 em que 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

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

outras regras, até mesmo a menor mudança pode afetar a efetividade do código (WALTON, 2015).

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 código 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 sob 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 na 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 serão apresentados 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>

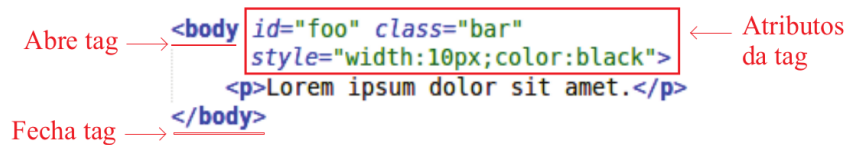
    <div id="div-bar" style="background-color:black; color:white;">
      <h2 class="header">Bar</h2>

      <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 repre-

Figura 2 – Estrutura de uma *tag*.

Fonte: Próprio autor

sentam a identificação única do elemento, uma meta informação identificando a sua utilidade e algumas regras de 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 utilização das *tags* `<style>` e `<script>`.

```
<style>
  body {
    background-color: yellow;
  }
  p {
    color: blue;
  }
</style>

<script>
  function myFunction (argument) {
    document.getElementById('foo').innerHTML = 'Hello JavaScript!';
  }
</script>
```

Fonte: Próprio autor

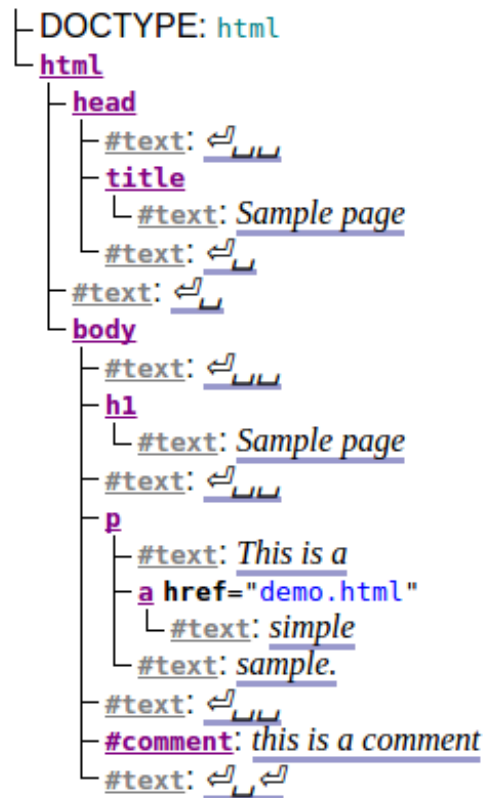
Além da definição *embedded*, pode-se definir estilos em CSS de outras duas formas: utilizando declarações *inline*, através do atributo `style` da *tag* HTML, ou referenciando a partir de um arquivo externo (*linked*), através do atributo `href` do elemento `<link>`.

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 e 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 (*children*) e os nós que se encontram no mesmo nível são chamados de irmãos (*siblings*). Os nós denominados `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 com as seguintes regras:

```
body p {
  color: blue;
  padding: 1px;
  background-color: gray;
}
```

Esta regra é destacada por um retângulo vermelho e rotulada com R_i e uma seta vermelha apontando para ela.

```
div > ul li {
  text-decoration: none;
  list-style: none;
}
```

Esta regra é rotulada com S_i e uma seta vermelha apontando para o seletor `div > ul li`.

```
#tabelaInicial > tr {
  font-weight: 100;
  font-family: Arial;
}
```

Esta regra é rotulada com P_i e uma seta vermelha apontando para a propriedade `font-weight`, e com V_i e uma seta vermelha apontando para o valor `100`.

```
.titulo {
  color: white;
  background-color: black;
  font-size: 25px;
}
```

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 do 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 (~) (ÇELIK et al., 2009).

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 (ÇELIK et al., 2009).

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-elementos) 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 em linguagens Orientadas a Objetos, que é alvo de estudos de Linguagens de Programação e Engenharia *Software*. A partir da perspectiva de trabalhos relevantes na área de conhecimento da qualidade de código fonte, foi 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 este 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), e as características apontadas neste trabalho como principal argumento, pretende-se identificar os atributos da linguagem que mais impactam na manutenibilidade de código CSS.

¹ Extensible Stylesheet Language

4 Metodologia

O primeiro passo da execução do trabalho consistiu na conceituação de manutenibilidade em código CSS. Esse objetivo foi alcançado por meio de referências bibliográficas e de uma pesquisa do tipo *survey* aplicada a desenvolvedores que escrevem código CSS em seu dia a dia, com níveis de experiência variados. Essa pesquisa pretendeu 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, foram identificados 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, foram determinados os pesos de cada critério, para a execução dos testes individuais das folhas de estilo. Nesta etapa, foi obtido um resultado numérico representando o cálculo dos pesos para os critérios encontrados no código CSS.

A partir dos valores encontrados nos testes, foram identificadas as relações entre o índice proposto e a quantidade de falhas identificadas em sistemas *web* de código aberto. Utilizando uma base de projetos de *software* de código aberto, foi feita uma análise e a referência cruzada, entre o valor obtido pela métrica do código CSS e o número de problemas reportados no projeto relacionados a alterações em código CSS.

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, foi aplicado um questionário exploratório, com o objetivo de identificar os conceitos do CSS consideradas centrais para a associação de qualidade do código.

4.2 Proposta da Métrica

As métricas de qualidade de código são indicadores numéricos baseados em características das linguagens de programação. Para este trabalho, essas características foram definidas a partir dos resultados obtidos no questionário. A métrica de qualidade para código CSS aqui

descrita, foi calculada a partir da presença e frequência de alguns dos recursos da linguagem, identificados pelos desenvolvedores no questionário, considerados impactantes para a manutenibilidade do código.

4.3 Avaliação dos Resultados

A partir da métrica proposta, foi desenvolvida uma ferramenta automática de cálculo da métrica. O programa lê um arquivo CSS, identifica as regras definidas e, a partir da definição proposta, calcula o valor obtido por este arquivo. Além do arquivo CSS, o arquivo HTML que o inclui também foi considerado para o cálculo.

Com os valores da métrica calculada, foi testada sua aderência a um projeto de código aberto, utilizando como indicador de manutenibilidade o número de defeitos cadastrado na ferramenta de controle de tarefas do mesmo. Utilizando de versões antigas, fez-se um histórico de modificação da métrica e o número de defeitos cadastrados, com o objetivo de avaliar o comportamento do indicador de manutenibilidade com os resultados obtidos no projeto ao longo do tempo.

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, pôde-se analisar os pesos de cada aspecto de qualidade do código CSS em função de sua manutenibilidade. Identificando as ocasiões em que se encontram maior ocorrência de efeitos colaterais, técnicas de organização das regras que colaboram para a legibilidade do código CSS e quais as características da linguagem que impactam, negativa ou positivamente, na sua manutenibilidade, com o objetivo de determinar os critérios que irão compor a métrica proposta.

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 foi 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 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, através de uma série de questões exploratórias, para identificar os aspectos mais abstratos de qualidade, e questões específicas, para identificar os aspectos estruturais da linguagem que representam maior dificuldade no período de manutenção. Para identificar a relevância das respostas do questionário, os respondentes foram agrupados por nível de proficiência.

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

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.

Para esta classificação, utilizou-se os critérios apresentados no Quadro 1, que indica as funcionalidades do CSS quanto ao seu nível de proficiência. Essa categorização foi feita pelo autor com base na complexidade e na frequência de uso de certas funcionalidades da linguagem CSS.

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.

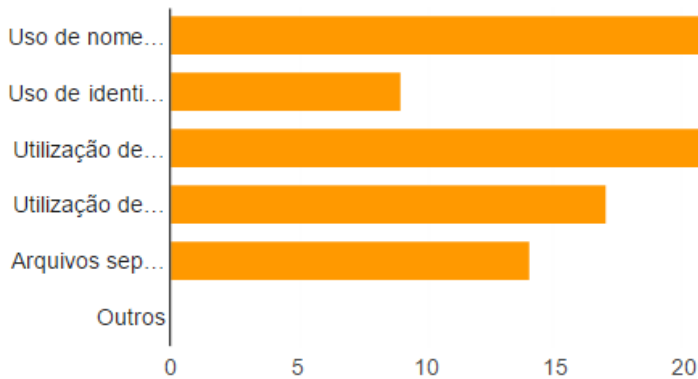
As questões propostas para identificar os aspectos de qualidade do código CSS identificaram resultados diversos. Como na Figura 7, pode-se verificar que mais de 90% dos respondentes identificaram os elementos estruturais, a nomenclatura coerente para as classes e identificadores (*id*) como sendo imprescindíveis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

Essas 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, pode-se 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.*

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

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.

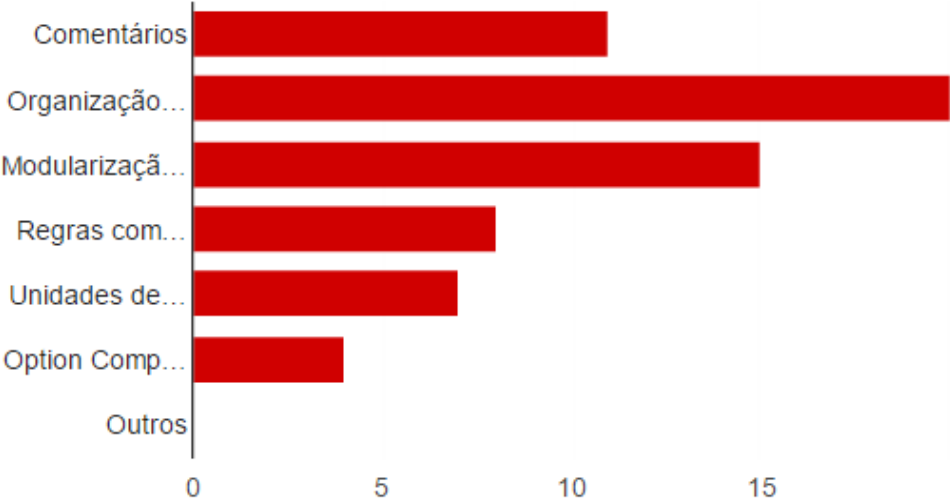
Ainda sobre os efeitos colaterais, foi elaborada uma questão com objetivo de identificar quais são as propriedades do CSS que mais os causam. As respostas foram variadas e por isso não foi possível identificar um padrão a partir desta pesquisa. No entanto, as respostas com elementos semelhantes sempre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

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?

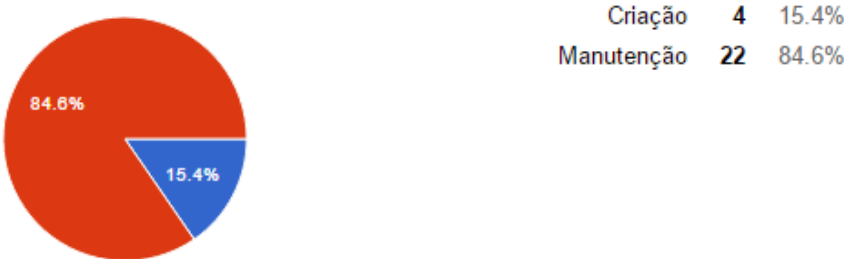


Comentários	11	42.3%
Organização dos seletores em seções	20	76.9%
Modularização das regras	15	57.7%
Regras complexas	8	30.8%
Unidades de medida flexíveis (% , vw , vh , etc.)	7	26.9%
Option Compactação das regras (agrupamentos, utilização de atalhos)	4	15.4%
Outros	0	0%

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	Corroborar	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

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.

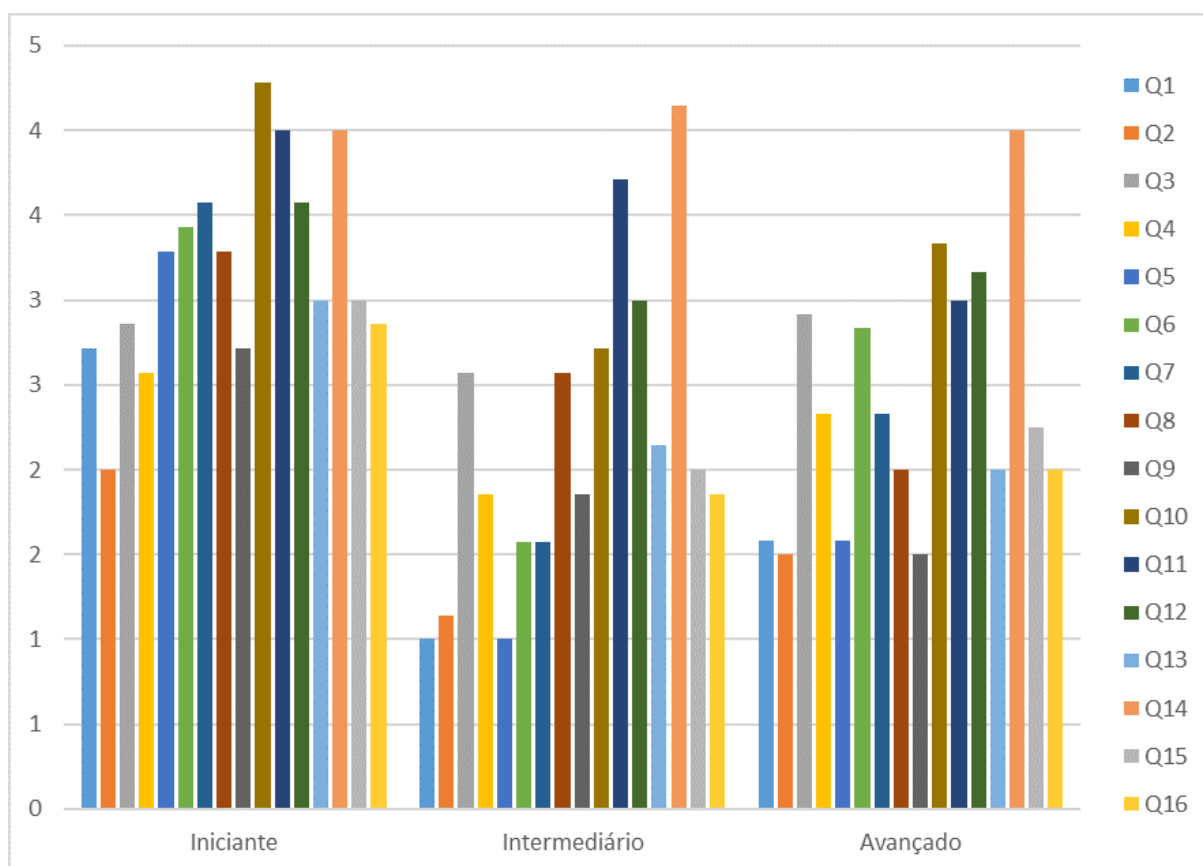
Essas respostas foram de suma importância para identificação de quais os critérios que deveriam, ou não, ser considerados para a avaliação da folha de estilo. Depois de identificados esses critérios, foi feita a análise das questões com exemplos de código, com o objetivo de definir os seus pesos.

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 em que precisariam modificar um trecho 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 dos valores atribuídos pelos respondentes para cada resposta, para determinar o grau de dificuldade de alterar os 16 trechos de código CSS representando as diferentes características da linguagem. Pode-se notar na Figura 10 a diferença de dificuldade encontrada pelos respondentes com diferentes níveis de proficiência.

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



Fonte: Próprio autor

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 foram definidos a partir da média ponderada de cada questão, para evitar 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 12 aspectos da linguagem a serem avaliados e, a partir deles, foram elaborados os critérios utilizados para o cálculo da métrica. O peso para cada um dos critérios de avaliação foi, então, definido a partir dos resultados obtidos no questionário, como visto na Tabela 1. Esses pesos

foram utilizados para o cálculo de cada métrica de forma individual, aplicando uma forma de avaliação correspondente a cada um dos critérios.

Tabela 1 – Tabela com peso de cada critério avaliado.

Critério	Peso
Seletores raros: {[^=], [\$=], ~, +, >}	3
Agrupamentos	2,8
Aninhamento	2,8
Propriedade simplificada	3,2
Pseudo elementos	2,8
Seletor com mais de 35 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

5.3.1 Identificação dos Critérios de Avaliação

Cada uma das 16 questões teve o objetivo de avaliar um aspecto da linguagem com o intuito de direcionar a escolha dos critérios de avaliação. Portanto fez-se um mapeamento das funcionalidades abordadas em cada questão e, a partir disso, determinou-se os critérios que seriam avaliados pelo *script* de cálculo da métrica.

Como pode ser visto no Quadro 3, as funcionalidades abordadas por cada uma das questões específicas foram definidas pelo próprio autor, de acordo com a sua experiência em codificação CSS, como sendo as funcionalidades que impactam, de alguma forma, na manutenibilidade do código.

A partir dessas funcionalidades foram identificados quais os métodos de avaliação caberiam a cada um deles. Esses métodos foram denominados critérios, que, para o *script* de cálculo, representam um padrão a ser identificado, contado e calculado. Desta forma foram definidos e construídos os critérios listados na Tabela 1.

Cada um dos critérios criados representa um impacto na qualidade e esse impacto foi definido como sendo o peso do critério avaliado. Dessa forma, o *script* avalia a presença ou não de cada critério nas regras definidas na folha de estilo CSS e, então, para cada ocorrência, soma o valor calculado para o critério ao total da métrica. Pode-se considerar como sendo a forma geral o que se define na Equação (2):

$$\text{critério(regra)} \rightarrow \# \text{ocorrências} * \text{peso} \quad (2)$$

Quadro 3 – Quadro com as funcionalidades exploradas em cada questão do questionário no Apêndice A

Questão	Funcionalidade
Q1	Seletor de classe simples
Q2	Seletor de id simples
Q3	Pseudo seletor de substring (^=)
Q4	Agrupamento de seletores
Q5	Pseudo classe
Q6	Propriedades simplificadas
Q7	Pseudo elemento
Q8	Seletor muito longo
Q9	At-rule
Q10	Media queries
Q11	Seletores com webkit, utilizando propriedades de webkit
Q12	Uso da clausula not
Q13	Child selector
Q14	Seletor de alta complexidade: localização + sibling + seletor universal
Q15	Conflito de herança na cor da tag b
Q16	Seletor de localidade: first-child

O valor encontrado pela métrica será então o somatório dos valores de todos os critérios encontrados. Como pode ser visto na Equação (3):

$$Métrica \leftarrow \sum_{i=1}^{12} critério_i(regra) \quad (3)$$

Em seguida foi definido como cada critério impacta na manutenibilidade, explicando em detalhe o funcionamento de cada um.

5.3.1.1 Seletores Raros

Os seletores definidos como raros, para este critério, foram identificados a partir da experiência do autor. Os seletores escolhidos foram: o seletor de atributo (`[attr=value]`); de descendente direto (`>`); de elementos irmãos (`~`) e o de primeiro elemento irmão (`+`). O uso destes seletores nas regras CSS podem dificultar na manutenção, pois podem não ser de fácil entendimento, aumentando a complexidade do código e impactando assim na manutenibilidade.

A participação destes seletores na métrica foi calculada a partir da regra geral, identificando a repetição deles em uma única regra e a frequência ao longo da folha de estilo. Nos testes executados, estes seletores tiveram pouca participação na métrica, o que corrobora com a premissa de que estes seletores são raramente utilizados.

5.3.1.2 Agrupamento de elementos

O agrupamento de seletores é identificado por regras que são aplicadas a elementos distintos. Ele é representado pela separação por vírgulas e é muito utilizado pelos desenvolvedores. Para o cálculo desta métrica, foi feito, então, uma contagem de seletores separados por vírgula para cada regra avaliada, porém, identificou-se que há um limite no impacto que um agrupamento muito extenso pode causar.

Devido ao limite identificado, foi utilizada uma função que mapeia o valor do critério em função do número de seletores agrupados na regra. Como pode ser visto na Equação (4), o limite de seletores agrupados é definido pelo denominador da função arco tangente, parâmetro este que foi ajustado para adequar aos testes executados.

$$\text{critério}(regra) \rightarrow \arctg\left(\frac{\#ocorrências}{20}\right) \quad (4)$$

5.3.1.3 Seletores Aninhados

Este critério foi desenvolvido para avaliar a profundidade dos seletores. Para tanto, foi necessária a contagem de espaços em branco entre seletores simples. Esse tipo de aninhamento pode ocorrer em várias combinações de seletores, *e.g.*, durante o agrupamento de seletores pode ocorrer um seletor aninhado.

Para o cálculo de impacto deste seletor também foi utilizada a Equação (4), considerando-se que existe um limite de impacto para a profundidade de um seletor. Após o cálculo individual de cada seletor, foi feita a soma de ocorrências durante a folha de estilo.

5.3.1.4 Propriedades Simplificadas

A simplificação das propriedades CSS são um atalho, que permitem que o desenvolvedor aplique todos os possíveis atributos a uma propriedade sem ter que declarar todos. Como pode ser visto no exemplo da Figura 11, não há uma sequência lógica fácil de se identificar para a declaração dos valores dessas propriedades, o que pode dificultar na manutenção desse tipo de regra.

Para calcular este critério foi identificadas as propriedades que possuíam valores atribuídos separados por espaço. Para cada propriedade, contou-se o número de ocorrências e somou-se ao final, seguindo o mesmo cálculo da Equação (2).

5.3.1.5 Pseudo Elementos

Os pseudo elementos podem ser utilizados para acessar propriedades especiais dos elementos HTML. Para este critério, foram considerados os pseudo elementos e as pseudo classes em um único critério, uma vez que obtiveram a mesma média de dificuldade. Das pseudo

Figura 11 – Exemplo de propriedades simplificadas.

```
<style>
  body {
    background: #ffffff url("img_tree.png") no-repeat right top;
    font: italic bold .8em/1.2 Arial, sans-serif;
    margin: 10px 5px 10px 5px;
  }
</style>
```

Fonte: Próprio autor

classes foram excluídos a cláusula `not` e os seletores de localidade, que receberam um critério de avaliação separado.

O cálculo foi feito, então, de acordo com a ocorrência de pseudo elementos e pseudo classes presentes no seletor das regras, novamente avaliando em qualquer situação que esses possam aparecer. A participação deste critério é medida como a frequência de ocorrências durante a folha de estilo.

5.3.1.6 Comprimento de Seletores

Para o cálculo do critério de comprimento dos seletores, 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. Nessa pesquisa, é apresentada uma distribuição do comprimento do seletor em caracteres.

A distribuição na Figura 12, mostra que a moda dos comprimentos encontrados pela pesquisa é em torno de 20 caracteres. A decisão tomada para o valor de ativação do critério foi então definida como 35 caracteres, um valor maior que a moda e que possui uma fatia expressiva na distribuição apresentada.

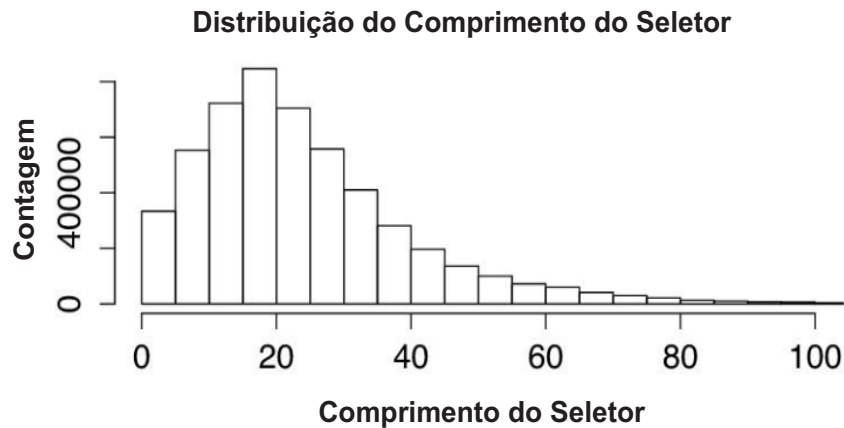
5.3.1.7 *At-rules*

Este critério avaliou as *At-rules*, denominadas assim por serem regras precedidas pelo símbolo `@` (arroba). Esse critério foi calculado utilizando-se a Equação (2), contando o número de repetições ao longo da folha de estilo. As *At-rules* podem definir uma série de atributos (*e.g.* `charset`, `import`, `namespace`, etc.), que dão suporte à importação de arquivos externos ou o poder de controle condicional, como no caso do `@media`.

5.3.1.8 *Media Queries*

A *At-rule* `@media` é um caso especial do CSS: uma declaração que permite ao desenvolvedor implementar uma regra condicionada ao argumento chamado *media query*. As

Figura 12 – Distribuição de tamanho dos seletores



Fonte: Traduzido de McPherson (2014)

media queries são condicionantes que avaliam aspectos da janela de renderização do HTML, ou informações do dispositivo no qual o HTML está sendo renderizado. As *media query* adicionam um nível a mais de complexidade no código CSS e têm ganhado espaço nas folhas de estilo devido à necessidade de estilos responsivos para as páginas *web*.

As regras `@media` podem conter várias regras CSS condicionadas a suas *media queries*, o que torna a avaliação mais complexa. Para o cálculo desse critério, foi feita uma análise do número de *media queries* presentes no arquivo CSS, cada uma avaliada como o peso determinado a elas na Tabela 1 e somadas ao valor final da métrica.

5.3.1.9 Prefixos

Os prefixos são utilizados para explicitar como a máquina de renderização do navegador irá interpretar uma propriedade CSS. Apenas algumas propriedades têm a necessidade de se utilizar esses prefixos. Mas quando utilizados, eles resultam em uma repetição de propriedade, o que implica numa dificuldade de leitura e manutenção do código maior.

Esse critério fica, então, responsável por identificar o número de repetições destes prefixos e somar o peso de cada uma delas ao resultado final da métrica.

5.3.1.10 Cláusula `:not`

O critério de avaliação da cláusula `:not` foi separada das pseudo classes por determinar um fluxo lógico completamente diferente dos seletores em geral. Essa cláusula determina que o seletor que ele sucede não pode existir na árvore de seleção do DOM, *i.e.*, serão selecionados os elementos que não possuírem o atributo negado em sua árvore DOM.

Essa inversão lógica pode causar um aumento na complexidade e deve ser, portanto, avaliada de forma exclusiva. Isto também pode ser identificado na Tabela 1, uma vez que o peso médio deste critério foi maior que o peso médio dos pseudo elementos e pseudo classes.

Esse critério calcula o número de ocorrências da cláusula `:not`, utilizando a forma geral e somando ao resultado final da métrica.

5.3.1.11 Complexidade do Seletor

Este critério foi calculado de forma diferente: ele identifica uma combinação de seletores e calcula, a partir da presença dos símbolos `+`, `~` e do identificador de propriedades `[attr=value]`, o valor da contribuição do critério como sendo o peso na Tabela 1 potencializado pela quantidade de vezes que estes elementos aparecem.

$$\text{critério}(\text{regra}) \rightarrow \text{peso}^{\# \text{ocorrências}} \quad (5)$$

Como pode ser visto na Equação (5), o critério de complexidade tem crescimento exponencial, fazendo o dele o critério que deve se ter mais atenção. A complexidade dos seletores cresce de acordo com a necessidade de regras que não contam com os atributos `id` ou `class` do HTML, devido à dificuldade de se encontrar os atributos desejados dentro do DOM.

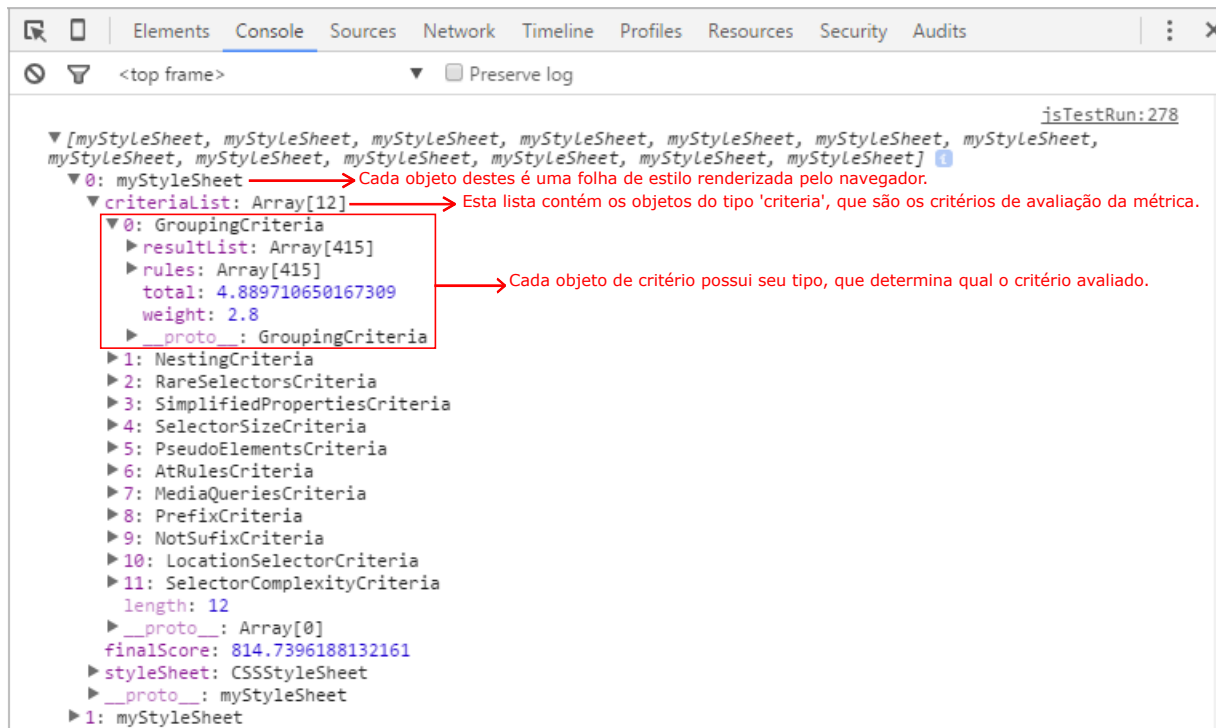
5.3.2 Script de Cálculo Automático da Métrica

Para fazer o cálculo da métrica de maneira automatizada, 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, a versão de implementação do ECMAScript do navegador deve ser levada em conta. Neste trabalho, foi utilizada a versão 6 do ECMAScript, utilizando de algumas funções do JavaScript que não estão disponíveis em todos os navegadores.

Foram necessários alguns testes do *script* para ajustes dos pesos e eles foram executados em várias páginas da *web*.

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. Dessa 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* é 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 13, pode-se visualizar a estrutura do resultado de uma execução do *script*. O código fonte desse *script* está disponível em um repositório no serviço gratuito de hospedagem

Figura 13 – Exemplo de resultado da execução do *script* de cálculo da métrica.

Fonte: Próprio autor

de código aberto GitHub¹. O objetivo final deste trabalho é gerar um *bookmarklet*² 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 um projeto em que o estilo da página fosse codificado em arquivos CSS, eliminando a possibilidade de aplicar os testes em projetos que utilizem preprocessadores³ CSS. Além desta limitação, deveria ser possível analisar um outro indicador de manutenibilidade de código, como tempo de evolução, número de defeitos ou algum índice de retrabalho.

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

² Pequena aplicação construída em JavaScript para ser executada a partir do atalho de favoritos no navegador. <<https://en.wikipedia.org/wiki/Bookmarklet>>

³ 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 como objeto de estudo para avaliação da métrica por atender a todas as limitações identificadas. Ele é um projeto maduro, largamente utilizado e com uma comunidade de desenvolvimento ativa. Pode-se ver na Figura 14 a quantidade de adições e exclusões de linha de código por semana, desde o ano de 2007. Ele também foi escolhido como objeto de estudo por utilizar somente código CSS para a construção de suas folhas de estilo.

Figura 14 – 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, foram utilizadas doze 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, tendo essa escolha sido feita devido à disponibilidade e possibilidade de identificar em qual ponto no tempo elas foram construídas, conforme exibido no Quadro 4.

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

Quadro 4 – Tabela com versões utilizadas e o número de defeitos gerados em cada uma delas.

Versão	Data de Lançamento	Número de Defeitos Gerados
1.369	31/07/2010	10
1.395	22/01/2011	8
1.423	25/07/2011	14
1.450	30/01/2012	13
1.475	01/08/2012	10
1.500	26/01/2013	13
1.525	29/07/2013	20
1.549	26/01/2014	22
1.574	27/07/2014	45
1.598	25/01/2015	23
1.622	27/07/2015	7
1.633	11/10/2015	2

Fonte: Próprio autor

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 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 em teste no momento.

O filtro do JIRA foi configurado utilizando a ferramenta de busca avançada, onde a busca poderia ser feita em forma textual. O filtro foi construído com os filtros disponíveis pelo JIRA, buscando dentro do projeto Jenkins pelos defeitos que possuísem a etiqueta 'css' ou a palavra CSS dentro do texto de sua descrição. Dessa forma, foi possível identificar quais os defeitos do projeto estavam relacionados de alguma forma com CSS, mesmo que tivessem sido encontrados em outro lugar e possuísem algum detalhe de CSS relacionados.

O alcance dos defeitos foi escolhido a partir de suas datas de criação: os criados a partir da versão em teste até a data de lançamento da versão que seria testada a seguir, no caso da versão mais atual foi feito de sua data de lançamento até o dia em que a pesquisa foi realizada. O filtro foi construído de maneira textual, utilizando a sintaxe do JIRA, e se parece com o seguinte: `project = JENKINS AND issuetype = Bug AND (labels = css OR text CSS) AND (createdDate >= "2010/07/31" AND createdDate < "2011/01/22") ORDER BY created ASC`. Os resultados dessa busca podem ser vistos no Quadro 4.

Com estas informações, é possível identificar uma relação entre o valor da métrica e o número de defeitos de cada versão e, a partir disso, averiguar se o valor da métrica proposta para um arquivo CSS (ou conjunto de arquivos) indica o nível de manutenibilidade desse(s) arquivo(s).

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

6.2 Dados para Teste

A partir do repositório de versões³ do Jenkins, selecionou-se algumas de forma a cobrir uma janela de tempo semestral para as avaliações.

Instâncias do Jenkins foram configuradas e executadas localmente para cada versão selecionada e, para cada uma, foi executado o *script* de cálculo automatizado da métrica para identificar o valor. Então, o valor encontrado de cada versão foi transcrito para uma planilha, 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 sendo testada no momento e a seguinte.

O valor da métrica foi então calculado como sendo o somatório dos resultados de todos os critérios de cada arquivo. Para os testes na página principal da aplicação, identificou-se mais de um arquivo que compunham o estilo da página. Junto a esses arquivos encontrou-se arquivos de uma biblioteca de componentes chamada YUI⁴ que é formada por arquivos JavaScript e CSS. Os arquivos pertencentes a essa biblioteca influenciam os resultados das métricas e certamente na manutenibilidade do CSS da aplicação.

Tabela 2 – Tabela com os resultados do *script* de cálculo automático para a versão 1.369

Critério	style.css	container.css	yui-skin.css	yui-container.css	yui-menu.css
Grouping	2,4614	0,6994	33,2557	1,2575	2,0847
Nesting	16,8729	6,6957	308,5058	12,2259	11,2661
Rare Selector	18,0000	0,0000	9,0000	0,0000	9,0000
Simplified Properties	0,0000	0,0000	0,0000	0,0000	0,0000
Selector Size	81,0000	39,0000	1881,0000	78,0000	90,0000
Pseudo Elements	61,6000	5,6000	123,2000	0,0000	5,6000
At Rules	0,0000	0,0000	0,0000	0,0000	0,0000
Media Queries	0,0000	0,0000	0,0000	0,0000	0,0000
Prefix	0,0000	0,0000	0,0000	0,0000	0,0000
Not Suffix	0,0000	0,0000	0,0000	0,0000	0,0000
Location Selector	8,4000	0,0000	0,0000	0,0000	0,0000
Selector Complexity	0,0000	0,0000	0,0000	0,0000	0,0000

Fonte: Próprio autor

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, como pode ser visto na Tabela 2. 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.

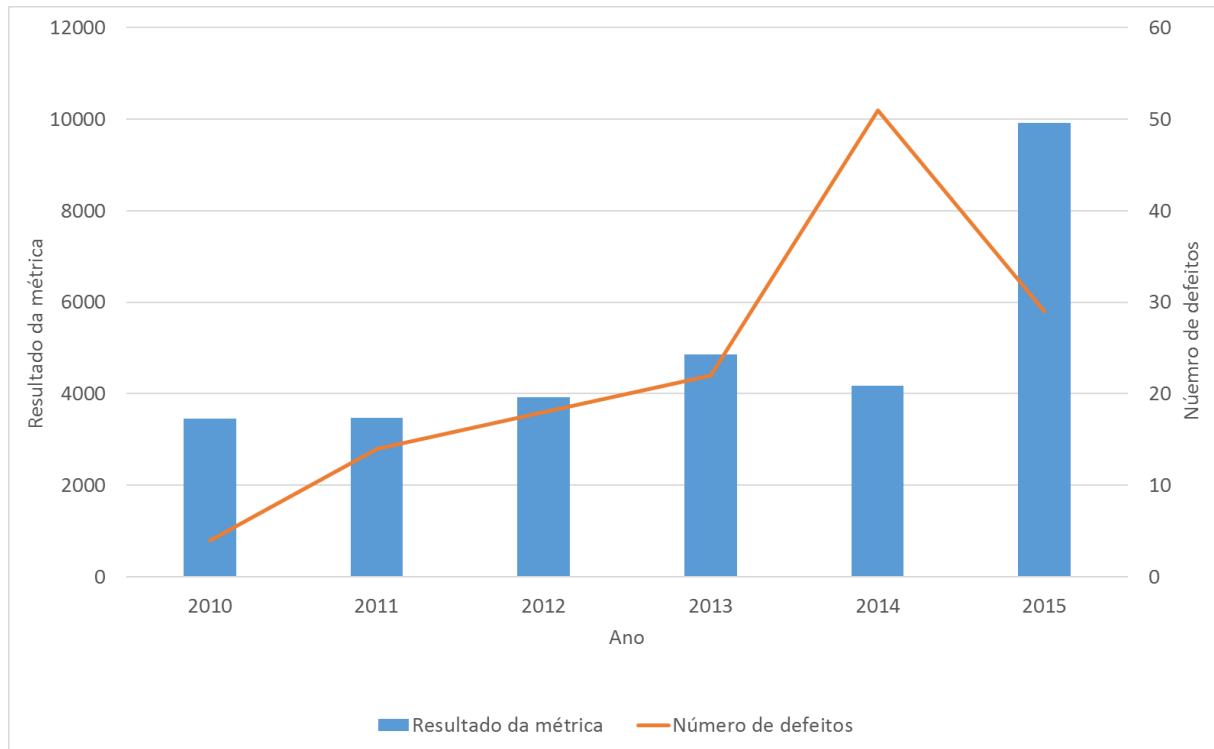
³ Disponível em <<https://updates.jenkins-ci.org/download/war/>>

⁴ Disponível em <<http://yuilibrary.com/>>

6.3 Resultados

Como as versões para execução do cálculo da métrica foram escolhidas com intervalos semestrais, considerou-se a média dos resultados como o valor médio para cada ano. Foi feito, então, um cruzamento do valor médio obtido para cada ano com o número de defeitos criados no mesmo ano.

Figura 15 – Comparação do resultado total da métrica em relação ao número de defeitos criados.



Fonte: Próprio autor

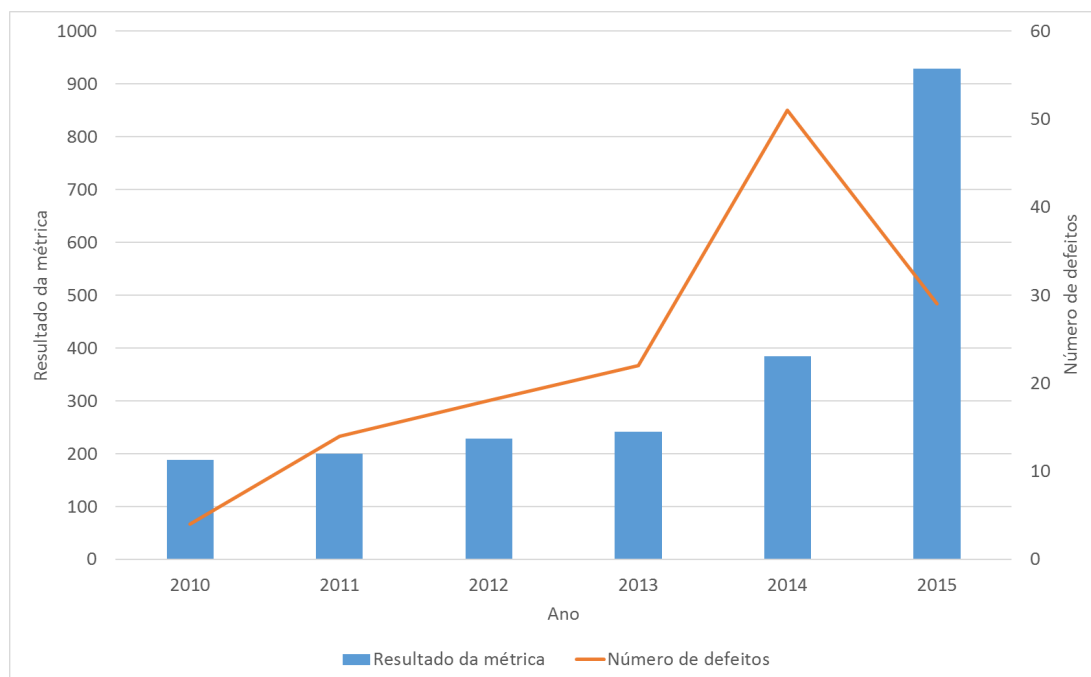
Pode-se notar na Figura 15 que o número de defeitos acompanha a progressão da métrica, mas o pico de cadastro de defeitos não se encontra no mesmo intervalo em que há um pico do resultado da métrica. Este comportamento pode ser explicado pelo fato do *layout* da aplicação ter sofrido uma mudança drástica entre as versões 1.549 e 1.574, disponibilizadas respectivamente em Janeiro/2014 (26/01/2014) e Julho/2014 (27/07/2014).

As folhas de estilo da biblioteca YUI representaram a maior porção do resultado total da métrica em grande parte das versões e também apresentou pouca mudança ao longo do tempo. Considerou-se, então, 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 que arquivos CSS haviam sofrido o maior número de alterações ao longo do tempo. Como pode ser visto na Tabela 3, o arquivo `style.css` foi, excepcionalmente, o que sofreu mais *commits*, sendo identificado, assim, como o arquivo de codificação CSS principal do projeto.

Tabela 3 – Número de *commits* para cada arquivo CSS renderizado na página principal do Jenkins.

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

A partir dessa 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.

Figura 16 – Comparação do resultado da métrica do `style.css` em relação ao número de defeitos criados.

Fonte: Próprio autor

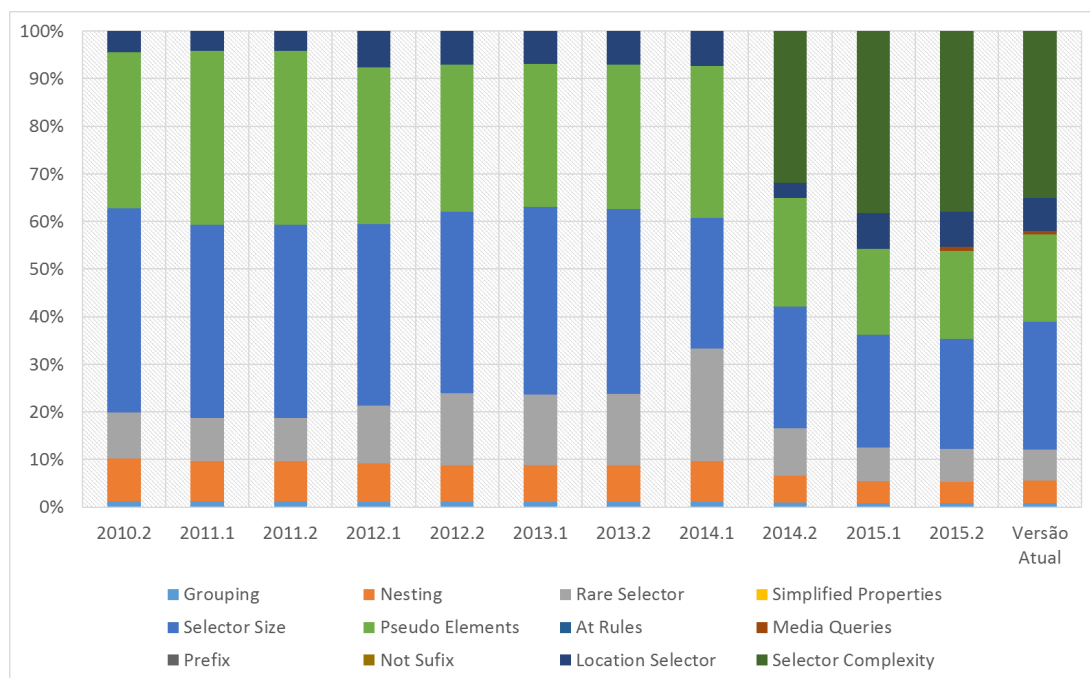
Pode-se notar na Figura 16 que a evolução da métrica foi crescente durante toda a progressão temporal e também um aumento considerável do resultado referente à mudança de *layout* em 2014. O comportamento do resultado da métrica para somente o arquivo `style.css` e o somatório de todos os arquivos (Figura 15) 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 à 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.

Essa 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 17, pode-se notar 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.

Figura 17 – Composição do valor da métrica por cada critério.



Fonte: Próprio autor

O aumento da distribuição dos critérios de avaliação no resultado da métrica, pode ser interpretado como um aumento na complexidade do código. Por meio da Figura 17, 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 16.

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.

7 Conclusão

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

A codificação de folhas de estilo CSS não é uma tarefa trivial e para se chegar ao resultado desejado é necessário um certo esforço de tempo. Apesar de aparentar simplicidade, a linguagem CSS contém algumas armadilhas para os desenvolvedores, como o efeito cascata, herança de propriedades e especificidade.

Devido a essa complexidade na codificação de CSS, a manutenção desse código se torna onerosa e é preciso identificar uma boa prática para organização do código, melhor utilização das propriedades e aspectos inerentes à linguagem.

Pode-se dizer que uma métrica de avaliação do código CSS será de grande valor para os desenvolvedores, auxiliando na identificação de possível causadores de erros, revisão de código e futuras otimizações nos processos produtivos. Esse valor somado à escassez de trabalhos de qualidade de código CSS, formam uma corrente em direção à pesquisa e desenvolvimento de uma métrica de qualidade, com a proposta de avaliar a manutenibilidade do código.

Este trabalho identificou critérios de avaliação e pontos de interesse, em direção à uma métrica de qualidade que viabilize o estudo sobre a etapa de manutenção de códigos CSS. Permitindo que próximas pesquisas apliquem testes para validar e ajustar os parâmetros aqui identificados.

A partir de uma pesquisa exploratória foram identificados alguns aspectos da linguagem CSS que compõem a qualidade de código, do ponto de vista de desenvolvedores que codificam CSS em seu dia a dia. A partir desses aspectos foi proposta uma métrica de qualidade para medir a manutenibilidade do código. Foi construído, então, um calculador automático, para avaliar o valor da métrica do código CSS de qualquer página *web*.

O calculador foi desenvolvido a partir de 12 critérios, identificados na pesquisa exploratória, e foi desenvolvido um *script* em JavaScript, para poder ser executado diretamente do navegador. Com o *script* pronto, foram feitos testes para ajustá-lo e prepará-lo para os testes no Jenkins.

Utilizando o *script*, foram feitos alguns testes na página principal da aplicação Jenkins. Esses testes foram feitos de para verificar a evolução do código CSS da aplicação ao longo do tempo, confrontando os resultados com o número de defeitos gerados, com o intuito de validar o resultado da métrica.

A partir da análise do histórico de *commits* dos arquivos CSS utilizados pela aplicação,

foi feita 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 exibia o mesmo perfil de evolução da métrica total. Pode-se entender que o comportamento dos defeitos está relacionado às modificações feitas no código CSS, uma vez que somente uma folha de estilo sofreu um grande número de alterações ao longo do tempo.

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

Este 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 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, foi feita uma adição importante na área de Engenharia de Software, principalmente no que diz respeito à interface *web*.

7.2 Trabalhos Futuros

Com o objetivo de dar continuidade a este 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.

Fazer modificações no arcabouço desenvolvido para que ele possa ser executado durante a fase de construção da aplicação, por exemplo, durante o processo de integração contínua, para que este processo não seja custoso à equipe de desenvolvimento.

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 11.

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 11.

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 13.

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 11.

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, 11 e 12.

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 11.

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 25 e 26.

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, 11 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%delimater"026E30F\\$nhhttp://dl.acm.org/citation.cfm?id=2493405](http://dl.acm.org/citation.cfm?doid=2493394.2493405%delimater)>. Citado na página 12.

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, 10 e 11.

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 12.

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 11.

W3C. **CSS**. 2015. Disponível em: <<http://www.w3.org/Style/CSS>>. Acesso em: 21 de maio de 2015. Citado na página 6.

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 3 vezes nas páginas 1, 2 e 12.

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 9 e 11.

ZUSE, H. **Software Complexity: Measures and Methods**. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1991. ISBN 0-89925-640-6. Citado na página 11.

ZUSE, H. **A Framework of Software Measurement**. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1997. ISBN 3110155877. Citado na página 11.

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 7.

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

1

2

3

4

Muito Complexo

5



22.

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

Muito Simples

1

2

3

4

Muito Complexo

5



23. .post-categories > li {

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

Muito Simples

1

2

3

4

Muito Complexo

5



24.

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

Muito Simples

1

2

3

4

Muito Complexo

5



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

