一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

CSS に対する自動リファクタリングに向けて

— W3C 勧告候補と草案に基づく CSS スメルの定義 —

大藤　陽斗†　　柗本　真佑†　　楠本　真二†

† 大阪大学大学院情報科学研究科
E-mail: †{hr-ohto,shinsuke}@ist.osaka-u.ac.jp

あらまし　CSS（Cascading Style Sheets）は HTML などの構造文章に対する修飾を定義する言語である．CSS はカスケードによる修飾ルールの合成やセレクタ設計の複雑さから保守が困難である．また CSS スメルの定義やリファクタリング手法も多数提案されている．しかし，CSS は 30 年経過した現在も新たな仕様が活発に議論されている言語であり，既存研究は現在の仕様を踏まえたスメル定義が十分でない．本研究の長期的な目的は，保守性の高い CSS ファイルの作成支援である．そこで，W3C 勧告候補と草案を対象として CSS スメルを定義する．本稿では CSS Variables と CSS Nesting に基づく CSS スメルを定義する．さらに，定義した CSS スメルの自動検出と自動リファクタリングに向けた課題を議論する．

キーワード　CSS，コードスメル，勧告候補，草案，リファクタリング，W3C

1. はじめに

CSS（Cascading Style Sheets）は HTML などの構造文章に対する修飾を定義する言語であり，Web が社会の情報インフラとなった現在では欠かせない技術である．文章構造とは独立にその修飾内容を定義することで，Web ページの保守性や再利用性が向上する．CSS により定義可能な修飾属性としては，背景色や文字色，フォントサイズ，レイアウトなどの静的な属性が代表的である．さらには，ボタン押下時のアニメーションや画面サイズ変更時のレイアウト変更といった，利用者の操作に対するインタラクティブかつ動的な修飾属性も定義可能である．Web の技術をネイティブアプリに組み込む WebView などのアイデアも多数登場している．すなわち，HTML と CSS は Web ページの記述という役割を超えて，アプリの汎用ユーザインタフェース定義としても活用され始めているといえる．

CSS で記述された CSS ファイル(注1)の保守やリファクタリングは，Web のフロントエンド開発において欠かせない工程である．CSS 自体は宣言的な仕組みであり，一般的な手続き型プログラミング言語のような条件分岐や繰り返しは存在しないものの，CSS 独自の難しさが多数存在する．例えば，セレクタと呼ばれる HTML 要素の選択の記述に対しては，クラス指定や ID 指定など様々な指定方法が存在しており，その適切な選択や組み合わせは容易ではない．具体的すぎるセレクタは HTML 構造の変更によって壊れやすく，単純すぎるセレクタはその文脈や意図の把握を困難とする．また CSS が「カスケード」である点も難しさの要因である．複数の修飾ルールを段階的に合成して全体の修飾内容を定義するため，個々の修飾定義が全体に与える影響の把握が困難である．これら CSS の難しさを低減するためには，適切な修飾を満たす CSS ファイルの記述のみならず，分かりやすく保守しやすい構造の保持と改善が重要である．また，CSS スメルの定義や自動リファクタリング手法も多数提案されている [1] [2] [3].

Gharachorlu が提案した CSS スメルの定義 [1] は 2026 年現時点での CSS の状況を十分に反映しているとは言い難い．CSS 自体は 1996 年にリリースされた古い言語ではあるものの，30 年経過した今なお新たな仕様が活発に議論されている “生きた” 言語である．CSS に加えられる新仕様は，CSS の修飾能力の拡張と CSS 内部の記述能力の拡張の 2 種類に大別できる．前者の一例は CSS Animations であり，従来 JavaScript で定義していた HTML 要素の動き（色の変化や物理位置の変更など）を CSS 単体で記述可能とする．後者の一例は CSS Variables であり，CSS ファイル内部で用いられる様々なプロパティ値（色やサイズなど）の変数化が可能である．CSS 内部の記述能力に対する様々な拡張により，より可読性と保守性の高い CSS ファイルの作成が可能となった．逆説的に，その新仕様を用いない箇所は保守性の低い記述箇所になったといえる．よって CSS に対する新たなコードスメルを検討する必要がある．

本研究の長期的な目的は保守性の高い CSS ファイルの作成支援である．そのために本研究では次の 3 つのトピックに取り組む．

- CSS に対する新たなコードスメルの定義
- CSS スメルの自動検出
- CSS スメルの自動リファクタリング

CSS スメルの定義に当たっては，Web の標準化団体 W3C

(注1)：本稿ではスタイルシート言語を CSS，CSS で記述された .css ファイルを CSS ファイルと区別して呼ぶ．

```css
body {
  background-color: #fcfcfc;
}
body div#main h1#title {
  color: #005a9c;
  font-size: 32px;
}
body h2 {
  color: #005a9c;
  font-size: 1.5rem;
}
```

図 1: CSS ファイルの一例

```html
<body>
 <div id="main">
  <h1 id="title">SIGSS</h1>
  <h2>研究会 開催情報</h2>
  <p>2025年7月研究会@札幌</p>
  <p>2025年12月研究会@姫路</p>
  <p>2026年3月研究会@対馬</p>
 </div>
</body>
```

(a) HTML ファイル

(b) レンダリング結果

図 2: 図 1 を適用する HTML ファイルとレンダリング結果

(World Wide Web Consortium) が公開している勧告候補と草案の 2 つを対象に検討する．勧告候補と草案は，仕様の最終段階である勧告の前段階ではあるものの，多くの Web ブラウザが実験的にその機能を取り込むケースが多い．すなわち勧告候補と草案まで進められた仕様は，Web フロントエンド開発者が既に利用可能な状態であるといえる．これらの仕様を対象とすることで，CSS の進化に伴い生まれた新たな CSS スメルの体系化を目指す．

本稿では 2 つの CSS 仕様を対象に，新たな CSS スメルの定義に取り組む．具体的には本稿執筆時点で勧告候補である CSS Variables，および草案である CSS Nesting の 2 つを対象とする．この 2 つの仕様はいずれも CSS 内部の記述能力の拡張であり，その不使用箇所は保守性や可読性の低下要因となり得る．また本稿では定義した 2 つの CSS スメルについて，どのように自動検出と自動リファクタリングを実現するかについて議論する．

2. 準　　備

2.1. CSS

CSS は HTML などの構造文章に対する修飾を定義するスタイルシート言語である．文章構造とは別に背景色や文字色，文字サイズ，レイアウトなどの修飾属性を定義することで独立性が高くなり，Web ページの保守性や再利用性が向上する．

図 1 と 図 2 を用いて CSS の記述方法と，CSS ファイルを HTML ファイルに適用したレンダリング結果を説明する．CSS ファイルは複数のルールセットで構成される． 図 1 の場合，3 つのルールセットが存在し，1 行目から 3 行目と 4 行目から 7 行目，8 行目から 11 行目が該当する．各ルールセットは修飾属性を適用する HTML 要素を指定するセレクタと，適用する修飾属性を設定する宣言ブロックから構成される．図 1 の場合 1 行目と 4 行目，8 行目がセレクタに該当し，2 行目と 5 行目から 6 行目，9 行目から 10 行目が宣言ブロックに該当する．セレクタは単純セレクタと複合セレクタ，複雑セレクタの 3 種類に分類される．単純セレクタはクラスや ID，要素名など HTML 要素の属性を 1 つだけ記述するセレクタである．複合セレクタは 2 つ以上の単純セレクタを空白を入れずに続けて記述するセレクタである．複雑セレクタは子孫結合子（半角スペース）や子結合子（>）などの結合子で単純セレクタまたは複合セレクタを結合して記述する．図 1 の 4 行目のセレクタ部分は，単純セレクタ `body` と，複合セレクタ `div#main` と `h1#title` を子孫結合子で結合した複雑セレクタである．この複雑セレクタは `body` 要素の子である `div#main` 要素の，さらに子である `h1#title` 要素を修飾対象として指定する．また宣言ブロックには修飾属性の項目を表すプロパティとそのプロパティ値で構成される宣言を複数記述する．1 行目から 3 行目のルールセットは 図 2(a) の `body` 要素を対象として背景色を指定しており， 図 2(b) の背景に反映される．4 行目から 7 行目のルールセットは 図 2(a) の `h1#title` 要素を対象として文字色と文字サイズを指定しており， 図 2(b) の見出しに反映される．8 行目から 11 行目のルールセットは 図 2(a) の `h2` 要素を対象として文字色と文字サイズを指定しており， 図 2(b) の小見出しに反映される．また 図 1 で指定していない修飾属性は， 図 2(b) の本文のように Web ブラウザの既定値で表示される．

2.2. W3C の標準化段階

W3C は CSS をはじめとする Web 技術の標準仕様を策定する国際的な非営利団体である．1994 年の設立以来，Web の長期的な発展を目指して相互運用性のある仕様の策定に取り組んでいる．

W3C が策定する仕様には，仕様の成熟度を示す標準化段階が付与されている．標準化段階は以下の順序で進められる．

（1）草案（Working Draft）
（2）勧告候補（Candidate Recommendation）
（3）勧告（Recommendation）

草案は仕様の設計段階であり，フィードバックに応じて内容が大きく変更される可能性がある．勧告候補は仕様のテスト段階であり，テストや実装を通じて仕様の検証が行われる．勧告は仕様の完成段階であり，十分な検証を経て安定した状態にあることを示す．

現在の CSS の仕様は，機能ごとに分割してモジュール単位で策定されている．2011 年 6 月に勧告に進められた CSS 2.1 まではモジュールという概念がなく，CSS は単一の仕様として策定されていた．CSS 2.1 を境に，CSS の仕様はモジュール単位での策定に移行した．各モジュールは CSS 2.1 仕様の

該当箇所を置き換え，または拡張する形で発展しており，モジュール化によって個々の機能の仕様が独立して策定されるようになった．

個々の機能の仕様が独立して策定されるようになり，CSS は急速に進化している．2026 年 1 月時点で，勧告や勧告候補である CSS モジュールの仕様は CSS 2.1 の仕様を含めて 46 個存在する．そのうち 26 個が 2015 年以降に初めて勧告候補に進められた CSS モジュールである．

2.3. CSS スメル

一般的な手続き型プログラミング言語にはコードスメルという概念が存在する [4]．コードスメルはソースコード中に含まれる可読性や保守性の問題に繋がる兆候であり，リファクタリングなどの改善作業によって解消されるべきとされている．C++ や Java，JavaScript などの手続き型言語ではコードスメルの定義や検出方法が多数提案されている [5] [6] [7]．

CSS においても，コードスメルの定義や検出方法が提案されている [1]．Gharachorlu は 8 種類のコードスメルの定義と検出方法を提案した．Gharachorlu が定義したコードスメルには，Too Much Cascading や Properties with Hard-Coded Values などがある． 図 1 の 4 行目のセレクタ部分は Too Much Cascading に該当する記述方法である．Too Much Cascading は過度に長いセレクタに関するコードスメルであり，文書構造がわずかに変更されただけでも定義した修飾属性の適用箇所が変化し，CSS ファイルの再利用性と保守性を低下させる要因になり得る． 図 1 の場合 ID セレクタ `#title` により対象要素を一意に指定できるため，`body div#main h1` は冗長でありセレクタを `#title` に簡約するべきである． 図 1 の 6 行目のプロパティ値は Properties with Hard-Coded Values に該当する記述方法である．Properties with Hard-Coded Values はプロパティ値として固定値を用いた宣言に関するコードスメルであり，画面サイズの変化や表示内容の変更などが軽微であってもレイアウトが崩れやすく，こちらも CSS ファイルの再利用性と保守性を低下させる要因になり得る． 図 1 の場合，プロパティ値を `2rem` のような相対値を用いて宣言するべきである．

2.4. 既存研究の課題

Gharachorlu の定義した CSS スメルは 2014 年の提案であり，2026 年の現時点で最新の CSS 仕様を反映していないという課題がある．CSS は，2011 年 6 月に CSS 2.1 の仕様が勧告されて以降も，2026 年の現時点に至るまで様々な機能拡張が続けられている．これらの機能拡張の中には，CSS が定義できるデザイン自体の表現能力の拡充のみならず，CSS 内部の記述能力の拡充も多数含まれる．前者の一例としては CSS Animations が挙げられる．CSS Animations は，HTML 要素の動き（色の変化や物理配置の変化など）を JavaScript を用いずに CSS 単体で記述できる機能を提供するモジュールである．後者の一例としては CSS Variables[(注2)][(注3)]が挙げられる．CSS Variables は CSS ファイル内で値の再利用を可能にし，変数のように値を扱える機能を提供するモジュールである．例えば，`:root { --main-color:red; }`のような特殊なルールセットを定義しておけば，同色を用いる箇所で `var(--main-color)` のように参照できる．

このような CSS 内部の記述能力の拡充によって，従来は記述が不可能であったより保守性の高い CSS ファイルを作成できる．言い換えれば CSS の仕様拡充に伴い，古い記述の一部が新たなコードスメル，すなわち保守性の低い箇所の候補となってきているといえる．よって新たな CSS スメルの定義，およびその検出方法とリファクタリング方法の検討の必要性が増してきた．

3. 研究のねらい

本研究の長期的な目的は，高い保守性を持つ CSS ファイル作成の支援である．そのために本研究では次の 3 つのトピックに取り組む．

- CSS に対する新たなコードスメルの定義
- CSS スメルの自動検出
- CSS スメルの自動リファクタリング

CSS スメルの定義に当たっては，本研究では W3C 勧告候補と草案の 2 つを対象に検討する．CSS は 2.2 節で述べたような標準化プロセスを経て，W3C により仕様が策定される．各種 Web ブラウザのベンダは W3C の公開する仕様に基づいて，Web ブラウザ内に新たな機能を組み込んでいく．この際，標準仕様である勧告の前の勧告候補や草案であっても，Web ブラウザへ実験的にその機能を取り込むことも多い．そのため，CSS ファイルを記述する開発者は勧告候補と草案の仕様に基づいた機能も利用可能である．例えば，2.4 節で説明した CSS Variables は 2026 年 1 月時点で勧告候補ではあるものの，Chrome や Edge，Safari，Firefox，Opera などの主要な Web ブラウザで利用可能である．よって，標準仕様として広く認知や普及が始まる前の仕様も対象として CSS スメルを定義すべきであると我々は考える．

本稿では 2 つの CSS 仕様を対象に，新たな CSS スメルの定義に取り組む．具体的には本稿執筆時点で勧告候補である CSS Variables，および草案である CSS Nesting[(注4)][(注5)]の 2 つのモジュールを対象とする．これら 2 つの仕様はいずれも CSS 内部の記述能力の拡張を目的としており，より可読性と保守性の高い CSS ファイルの作成が可能とする．逆説的に，その新仕様を用いない箇所は保守性の低い記述箇所になったといえる．よって CSS に対する新たなコードスメルを検討する必要がある．

続く 4 節では，本稿で対象とする 2 つの CSS モジュールを紹介する．さらに，5 節でそのモジュールの登場によって生まれる新たな CSS スメルの定義を検討する．6 節では今後の展望として，定義した CSS スメルの自動検出と自動リファクタリングの実現方法，および CSS スメルの拡充について述

(注2)：正式名称は CSS Custom Properties for Cascading Variables Module
(注3)：`https://www.w3.org/TR/css-variables-1/`
(注4)：正式名称は CSS Nesting Module
(注5)：`https://www.w3.org/TR/css-nesting-1/`

```css
:root {
  --main-bg-color: #fcfcfc;
  --head-fg-color: #005a9c;
}
body {
  background-color: var(--main-bg-color);

  #title {
    color: var(--head-fg-color);
    font-size: 2rem;
  }
  h2 {
    color: var(--head-fg-color);
    font-size: 1.5rem;
  }
}
```

図 3: CSS Variables と CSS Nesting を用いた図 1 と等価な CSS ファイル

べる．

4. 保守性を改善する CSS モジュール

本節では本稿で対象とする 2 つの CSS モジュールである CSS Variables と CSS Nesting を紹介する．両者はいずれも CSS ファイル内部の記述能力を拡張し保守性改善に繋がる新たな機能を提供するモジュールである．

4.1. CSS Variables

CSS Variables は，CSS ファイル内で再利用したい値をカスタムプロパティとして名前付きで定義し，`var()` 関数により任意のプロパティの値を参照可能とするモジュールである．CSS Variables の登場により，CSS ファイル内で繰り返し出現する特定の値を一箇所にまとめて管理できるようになった．CSS Variables は 2012 年 4 月に草案として公開され，2026 年 1 月時点で勧告候補のモジュールである．

図 3 は CSS Variables と 4.2 節で紹介する CSS Nesting を用いて図 1 をリファクタリングした CSS ファイルである． 図 3 の 1 行目から 4 行目では`:root` セレクタ内でカスタムプロパティを定義している．`:root` は疑似クラスと呼ばれる特殊なキーワードであり，HTML の全体構造を指し示している．一般的に，カスタムプロパティはこのクラス内で定義される．カスタムプロパティ名は`--`で始まり，任意の値を設定できる．図 3 では当該 Web ページ全体の背景色を`--main-bg-color`，見出しの文字色を`--head-fg-color` として定義し，6 行目および 9 行目，13 行目で `var()` を用いてプロパティ値を参照している．

CSS Variables を用いる利点は主に 2 つある．1 つ目は可読性の向上である．値に名前を与えることで意味が明確になる．2 つ目は保守性の向上である．同じ値の記述の繰り返しを避けることで，値の変更時に複数箇所を修正する必要がなくなる．

4.2. CSS Nesting

CSS Nesting はあるルールセットの内部に別のルールセットをネストする記述を可能にするモジュールである．CSS Nesting では内側ルールセットのセレクタを外側ルールセットのセレクタの子孫セレクタとして扱う．そのため従来はセレクタを連ねて記述していた対象の指定を，ネストして記述できるようになる．CSS Nesting によりセレクタの重複記述が減り，関連するルールセットをまとめて記述できるようになる．CSS Nesting は 2021 年 8 月に草案として公開され，2026 年 1 月時点においても草案のモジュールである．

4.1 節と同様，図 3 を用いて CSS Nesting を説明する． 図 3 の 5 行目では body セレクタを外側ルールセットのセレクタとして記述し，その内部の 8 行目から 11 行目および 12 行目から 15 行目で`#title` セレクタおよび `h2` セレクタのルールセットを記述している．これにより `body` セレクタの繰り返しが除去され，関連するセレクタの修飾定義を同じルールセット内に集約できる．

CSS Nesting を用いる利点は主に 2 つある．1 つ目は可読性の向上である．関連するルールセットをネストしてまとめて記述できるため，ルールセット間の関係が明示される．2 つ目は保守性の向上である．同じセレクタの記述の繰り返しを避けることで，セレクタを変更する際の修正箇所が減る．

5. CSS スメルの定義

4 節で述べた CSS モジュールの登場によって生まれる新たな CSS スメルを定義する．ここでは 2 つのモジュールに対してそれぞれ 1 つの CSS スメルを定義する．コードスメルとは保守性悪化の兆候であり，確実に悪化する要素であると断定できない．しかしながら，自動検出や自動リファクタリングなどの CSS スメル定義の活用先を考えるに当たっては，どの程度の兆候をコードスメルとして扱うかを定量的に定義する必要がある．

5.1. 変数の不使用

CSS Variables の登場を背景として，変数の不使用という CSS スメルを新たに定義する．変数の不使用は CSS ファイル内で意味が明示されていないリテラル値がプロパティ値として記述されている箇所を指す．このような記述はプロパティ値の意味が CSS ファイル上に現れず，可読性を低下させる．さらに同じ意味のリテラル値が複数箇所に散在する場合，値の変更時に修正漏れが生じやすく保守性を低下させる要因となり得る．

図 1 の 2 行目や 5 行目，9 行目のプロパティ値は，変数の不使用に該当する記述方法である．2 行目のプロパティ値は Web ページ全体の背景色という明確な意味を持つが，リテラル値として記述されている．また 5 行目や 9 行目のプロパティ値は見出しの文字色を指定しており同じ意味を持つ値であるにもかかわらず，`#005a9c` というリテラル値が繰り返し記述されている． 図 1 の場合， 図 3 の 2 行目から 3 行目のようなカスタムプロパティを定義し，6 行目や 9 行目，13 行目のように `var()` を用いて参照するべきである．

しかし，CSS ファイル内のあらゆるリテラル値をカスタムプロパティとして定義するべきではない．CSS には unset や none など，仕様で意味が定義されたキーワード値が存在する．また同じリテラル値であっても意味が異なる場合があるため，同じカスタムプロパティとして定義してしまうとかえって可読性や保守性を低下させてしまう．

したがって，本研究では以下のいずれかを満たす場合を変数の不使用と定義する．

・同じ意味のリテラル値が複数箇所に記述されている
・明確な意味を持つリテラル値が記述されている

2 つ目の条件が指す明確な意味を持つ値とは，Web ページ全体の背景色や基本となる文字色のように役割を言語化できる値を意味する．

一般的な手続き型言語において，意味が明示されていないリテラル値は Magic Number と呼ばれている [4] [8]．Magic Number は，リテラル値単体ではソースコード内における値の意味の把握が困難であり，可読性の低下を招き得る．つまり，値の意味が説明されないこと自体が問題であるため 1 回の出現であってもコードスメルとなり得る．どの程度の頻度や文脈をもとに検出すべきか一般化された閾値は確立していないため，本研究で採用した定義が妥当かどうかは実際の CSS ファイルを対象とした検証が必要である．

5.2. 浅い構造

次に，CSS Nesting に対応する CSS スメルとして浅い構造を定義する．浅い構造は共通の親セレクタを持つ複雑セレクタのルールセットが複数存在するにもかかわらず，それらがネストされず分散して記述されている箇所を指す．このような記述は同じ親セレクタが冗長に繰り返されるため，ルールセット間のまとまりが把握しにくくなる．また共通する親セレクタの変更時に複数箇所の修正が必要となり，修正漏れが生じやすく保守性を低下させる要因となり得る．

図 1 の 4 行目から 7 行目や 8 行目から 11 行目のルールセットは，浅い構造に該当する記述方法である．これらのルールセットは共通の親セレクタとして `body` セレクタを持つにもかかわらず分散して記述されている． 図 1 の場合， 図 3 のように `body` セレクタのルールセット内部に`#title` や `h2` セレクタのルールセットを集約して記述すべきである．

しかし，共通するセレクタを持つルールセットを常にネストするべきというわけではない．過度なネストは階層の追跡自体が困難になり，可読性を損なう恐れがある．よって，ネスト可能かだけでなくネストの深さが過度にならない場合に CSS スメルとして扱う必要がある．

したがって，本研究では以下の両方を満たす場合を浅い構造と定義する．

・共通セレクタを持つルールセットが複数存在する場合
・ネストの深さが 3 以下に収まる場合

一般的な手続き型言語において，ネストの深さは可読性に影響する要因として知られている [9]．深いネストはメソッドの全体構造の把握を妨げ，保守性を低下させる要因となり得る．そのため，ガード節の導入やメソッド抽出など多くのリファクタリング手法が提案されている [4] [10] [11]．一方，CSS は構造の浅さがコードスメルとなる．ネストを用いない平坦な構造では，関連するルールセット間の関係性が把握しにくいためである．ただし，過度なネストは手続き型言語と同様に，ネストの深さが問題となるため適切な深さの範囲でネストにする必要がある．本研究では，共通する親セレクタを持つルールセットが 2 つ以上存在し，集約後のネストの深さが 3 以下の場合を CSS スメルと定義したが，これらの閾値が妥当かどうかは実際の CSS ファイルを対象とした検証が必要である．

6. 今後の課題と展望

6.1. CSS スメル自動検出に向けて

今後の課題として，まずは定義した CSS スメルの自動検出の実現が挙げられる．いずれの CSS スメルにおいても，値や親セレクタの表層的な一致のみを根拠に検出すると役割の異なる記述を同一視して誤検出を招く恐れがある．そのため，プロパティ名やセレクタ名などの周辺情報を手がかりとして文脈を推定し，検出対象を選別する手法を検討する必要がある．

CSS Variables に対応する CSS スメルである変数の不使用においては，CSS ファイル内で繰り返し出現する特定のリテラル値が同じ意味で用いられているか否かの類推が課題となる．同じ値であっても意味が異なる場合があり，単純な値の一致だけで 1 つのカスタムプロパティに集約するのは不適切である．例えば同じ `100%` という値が画像の幅と文字サイズの両方に用いられている場合，両者は異なる意味を持つため別々のカスタムプロパティとして定義すべきである．手続き型言語においては，変数の使用文脈からその変数の意味を捉えて自然な変数名を推定する手法が提案されている [12]．CSS には型情報やデータフロー情報が存在しないため，プロパティ名やセレクタ名を文脈として用いて同じ意味か否かを判定する方針が考えられる．

また CSS Nesting に対応する CSS スメルである浅い構造においては，共通の親セレクタを持つ複雑セレクタのルールセットをネストの候補とすること自体は比較的容易である．しかし，複雑セレクタの共通部分のみを根拠にネストする候補とみなすと，意味的に関連性の低いルールセットを誤って集約する，あるいは過度なネストに繋がる恐れがある．手続き型言語においても意味的なまとまりの自動判定は困難な課題である．この課題に対し，依存関係や機能的関連性に基づいてコードのまとまりを判定する手法が提案されている [13] [14]．CSS ではセレクタ名の類似度やルールセット間の近接性，宣言するプロパティ集合の類似度といった指標を用いて集約すべきルールセットを選別する方針が考えられる．

6.2. 自動リファクタリングに向けて

CSS スメル自動検出の次なる課題として，検出した CSS スメルの自動リファクタリングの実現が挙げられる．いずれの

CSS スメルにおいても，機械的に変換するだけでは意図に反して可読性や保守性を損なう恐れがある．そのため，4.1 節で述べた変数の不使用ではカスタムプロパティ名の決定する手法，4.2 節で述べた浅い構造ではリファクタリング前後でのレンダリング結果の等価性を保証する手法の検討が必要である．

6.1 節の方法で検出された変数の不使用に対して，抽出した値に対する適切なカスタムプロパティ名の付与が課題となる．カスタムプロパティ名が不適切であれば，値に意味を与えるという利点を失い，かえって可読性を低下させ得る．手続き型言語では，変数の使用文脈から変数名を推定する手法 [12] や同じ種類の変数名の再利用と大規模コーパスから命名規則をマイニングして適用する手法 [15] が提案されている．CSS においては，プロパティ名や値の種類，セレクタ名に含まれる単語からカスタムプロパティ名を構成する方法と，多数の CSS ファイルからカスタムプロパティを収集し命名規則を導出して適用する方針が考えられる．

また 6.1 節の方法で検出された浅い構造に対しては，リファクタリング前後のレンダリング結果が等価であることを検証する手法が求められる．CSS Nesting を用いたリファクタリングではネストすることでセレクタが変化する．CSS では複数のルールが同一の HTML 要素に適用される場合，セレクタの変化によりリファクタリングの前後でレンダリング結果が等価ではなくなる場合があるため，等価性を検証する手法の確立が必要である．

6.3. CSS スメル定義の拡充

本稿では，CSS Variables および CSS Nesting の 2 つのモジュールに基づいて CSS スメルを定義した．しかし，CSS ファイル内部の記述能力を拡充する CSS モジュールは他にも多数存在する．その一例として CSS Color(注6)が挙げられる．近年 CSS Color の仕様拡張では，`color-mix()` という色を混合して新しい色を得る関数が導入された．`color-mix()` を用いると基準色から派生色を一貫して生成できるため，CSS ファイル内で近似色を複製・微調整する作業を削減できる．その結果として派生色の意図が明確になり，配色調整時の修正漏れや不整合の混入を抑制できる可能性がある．`color-mix()` に対しては基準色との関係を明示できるにもかかわらず，独立した色として記述されているという点が CSS スメルとして定義できると考える．

以上のように，CSS Variables や CSS Nesting 以外のモジュールに対しても当該機能が活用されていない記述箇所を CSS スメルとして定義できる可能性がある．今後はこれらのモジュールについても 5 節と同様の観点で検討を進め，CSS スメルの定義を拡充していきたい．

7. おわりに

本稿では保守性の高い CSS ファイルの作成支援を目的として，CSS の新仕様に基づく新たな CSS スメルの定義を提案した．また，定義した CSS スメルの自動検出手法および自動リファクタリング手法について検討した．

今後の課題の 1 つとして定義した CSS スメルの自動検出の実現が挙げられる．本稿で検討した方針に基づき，CSS ファイルを解析してスメルを検出するツールの実装に取り組む．また定義した CSS スメルを検出し，その妥当性を検証することも重要な課題である．実際の Web ページで使用されている CSS ファイルを対象に検出を行い，定義したスメルの妥当性を評価する．最後に検出した CSS スメルの自動リファクタリングの実現を目指す．本稿で検討したリファクタリング方針に基づき，CSS スメルを改善するツールの実装に取り組む．

謝辞 本研究の一部は，JSPS 科研費（JP25K15056, JP25K03102, JP24H00692）による助成を受けた．

(注6)：https://www.w3.org/TR/css-color-5/

文　献

[1] G. Gharachorlu, “Code Smells in Cascading Style Sheets : an Empirical Study and a Predictive Model,” Master’s thesis, University of British Columbia, 2014.

[2] L. Punt, S. Visscher, and V. Zaytsev, “The A? B* A Pattern: Undoing Style in CSS and Refactoring Opportunities it Presents,” Proc. International Conference on Software Maintenance and Evolution, pp.67–77, 2016.

[3] D. Mazinanian, N. Tsantalis, and A. Mesbah, “Discovering refactoring opportunities in cascading style sheets,” Proc. International Symposium on Foundations of Software Engineering, pp.496–506, 2014.

[4] M. Fowler and K. Beck, Refactoring: Improving the Design of Existing Code, Addison-Wesley Longman Publishing Co., Inc., 1999.

[5] T. Mashiach, B. Sotto-Mayor, G. Kaminka, and M. Kalech, “CLEAN++: Code Smells Extraction for C++,” Proc. International Conference on Mining Software Repositories, pp.441–445, 2023.

[6] N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur, “DECOR: A Method for the Specification and Detection of Code and Design Smells,” Transactions on Software Engineering, vol.36, no.1, pp.20–36, 2010.

[7] A.M. Fard and A. Mesbah, “JSNose: Detecting JavaScript Code Smells,” Proc. International Working Conference on Source Code Analysis and Manipulation, pp.116–125, 2013.

[8] W.C. Wake, Refactoring Workbook, Addison-Wesley Longman Publishing Co., Inc., 2004.

[9] M.H. Clifton, “A technique for making structured programs more readable,” ACM SIGPLAN Notices, vol.13, no.4, pp.58–63, 1978.

[10] D. Boswell and T. Foucher, The Art of Readable Code: Simple and Practical Techniques for Writing Better Code, O’Reilly Media, 2011.

[11] R. Saborido, J. Ferrer, F. Chicano, and E. Alba, “Automatizing Software Cognitive Complexity Reduction,” Journal on IEEE Access, vol.10, pp.11642–11656, 2022.

[12] B. Rohan, P. Michael, and S. Koushik, “Context2Name: A Deep Learning-Based Approach to Infer Natural Variable Names from Usage Contexts,” arXiv:1809.05193v1 [cs.SE], 2018.

[13] N. Tsantalis and A. Chatzigeorgiou, “Identification of Extract Method Refactoring Opportunities,” Proc. European Conference on Software Maintenance and Reengineering, pp.119–128, 2009.

[14] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, A. Gkortzis, and P. Avgeriou, “Identifying Extract Method Refactoring Opportunities Based on Functional Relevance,” Transactions on Software Engineering, vol.43, no.10, pp.954–974, 2017.

[15] T. Wang, H. Liu, Y. Zhang, and Y. Jiang, “Recommending Variable Names for Extract Local Variable Refactorings,” Transactions on Software Engineering and Methodology, vol.34, no.6, pp.1–38, 2025.