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

Arduinoスケッチの開発支援を目的とした Arduinoスメルの定義と検出ツールの試作

忠谷　晃佑†　　柗本　真佑†　　楠本　真二†

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

あらまし　安価で手軽な組込み用のマイコンプラットフォームとして，Arduino が普及している．Arduino の制御プログラムであるスケッチの記述においては，Arduino 固有の操作や Arduino 言語の実装に起因するミスが避けられない．本研究ではスケッチにおける典型的な誤りをコードスメルと捉え，Arduino スメルの定義とその検出に取り組む．Arduino スメルの定義においては，公式リファレンスや実験科目で学生が記述したスケッチから目視により典型的な誤りを探し出す．最終的に 7 種類のスメルを発見・定義し，Web 上に公開されている実際のスケッチ 331 個から合計 1,609 個のスメルの自動検出に成功した．

キーワード　Arduino，スケッチ，コードスメル，コードスメル検出

1. はじめに

安価で手軽な組込みシステム用マイコンボードプラットフォームとして，Arduino[1]が普及している．Arduino は小型マイコンと入出力ポートを備えたワンボードマイコンの一種であり，センサを用いた外界変化の感知や，アクチュエータを用いた外界への作用が容易に実現できる．ハードウェアとソフトウェアの両方がオープンソースであり，また安価かつ手軽という特性から，学習現場での活用 [1] のみならず，商用デバイスの試作 [2] にも広く用いられている．

Arduino ボードの制御に用いられる Arduino スケッチの記述においては，スケッチ固有のミスが避けられない．このスケッチは Arduino 言語と呼ばれる C++ のラッパー言語により記述される．スケッチでは Arduino 操作に特化した様々な関数が利用可能であるが，その利用の際には新たなライブラリや API の習得と同様，一定の難しさが発生する．また，Arduino 制御は根本的にリアルタイムシステムの制御であるという特性からも，典型的なバッチ処理とはその制御の考えが大幅に異なる．具体的には，ループ制御や割り込み制御が必要であり，その適切な制御のためには Arduino が搭載するマイコン（Atmel AVR や Arm Cortex-M4）の仕様に対する理解も必要となる．

多くのプログラミング言語では，開発者が陥りがちな典型的なミスを表す概念としてコードスメルが定義されている．スメルはソースコード中に含まれる保守性や可読性に繋がる問題の兆候であり，リファクタリング等の改善作業によって解消されるべきとされている [3]．C や Java，JavaScript などの言語ではスメルの定義や検出方法が多数提案 [4] [5] [6] されている．これらのスメル検出ツールを用いることで，各種言語における典型的な誤りを自動検出することが可能となる．

（注1）：https://www.arduino.cc

本研究では，Arduino スケッチに対する典型的なミスの低減を目指して，Arduino 言語に特化したコードスメル（Arduino スメル）の定義とその自動検出に取り組む．Arduino スメルの定義においては，Arduino 公式が公開しているリファレンスの記載内容，および初学者が作成した複数の Arduino スケッチから典型ミスを洗い出す．スメルの自動検出にはスケッチの抽象構文木に対する部分木マッチングを用いる．結果として 7 種類の Arduino スメルを発見・定義し，Web 上に公開されている実際のスケッチ 331 個から合計 1,609 個のスメルの自動検出に成功した．

2. 準　　備

2.1 Arduino

Arduino とは組込みシステム用マイコンボードである．Arduino ボードにはマイコンの他に汎用ピンが備わっており，それらにセンサやアクチュエータを接続して外界変化の取得や外界への作用が可能である．組込みシステムの学習にも利用されている [1] ほか，商用デバイスの試作に用いられている [2]．なお Arduino には汎用ピンの数やマイコンなどが異なる，数種類が存在する．本稿では Arduino ボードを ArduinoUno に限定する．

2.2 スケッチ

Arduino ボードはスケッチと呼ばれるプログラムにより制御される． 図 1 にスケッチの例を示す．このスケッチが書き込まれたボードの汎用ピン 2 番に LED を接続することで 1 秒ごとに LED が点滅を繰り返す．

スケッチは C++ のラッパー言語により記述でき，プログラマは Arduino 実行開始時に一度だけ実行される `setup()` 関数と `setup()` の実行後無限に繰り返される `loop()` 関数を記述する．プログラマはこれら 2 つの関数に Arduino 固有の関数や，マイコンの持つレジスタの操作を記述し Arduino ボードを制御する．

This article is a technical report without peer review, and its polished and/or extended version may be published eleswhere.
Copyright ©2024 by IEICE

```
const int ledPin = 2;
int time = 0;
bool ledState = false;
void setup() { // 実行直後1度だけ実行
  DDRB |= (1 << ledPin); // ピンを出力モードに
}

void loop() { // 無限に繰り返し
  if(millis() - time > 1000){ // 1000ms毎に
    if (!ledState)
      PORTB |= (1 << ledPin); // ピン出力を1に
    else
      PORTB &= (0 << ledPin); // ミス2
    time = millis(); // ミス1
    ledState = !ledState;
  }
}
```

図 1　1 秒ごとに LED を点滅させるスケッチの例

2.3　スケッチ記述特有の難しさ

スケッチの記述では他の言語には見られない固有のミスを犯しやすい．実際に 図 1 に示す例には 2 つのミスが含まれている．1 つ目は 14 行目において変数 `time` に `millis()` の返り値を格納している点である．ここで，`millis()` の返り値は符号なし 4 バイト整数（`unsigned long`）であるが，変数 `time` は符号あり 2 バイト整数型である．そのため 図 1 に示したスケッチは `time` の値が 32 秒（$32,768(2^{16-1}) \simeq 32,000$ ミリ秒）でオーバーフローする．このミスによってシステムに混入するバグはシステム実行後直ちに顕在化しない潜在バグ [7] [8] であり，プログラマにとって気が付きにくい．2 つ目はビット演算の誤りである．スケッチではレジスタの操作のためにビット演算が多く用いられるが，プログラマの意図と異なる演算を行う誤りが発生する． 図 1 の 13 行目では `PORTB` の 2 ビット目を 0 にしようとしている．しかしこの演算は 0 と `PORTB` のビット積を取る演算であり，意図せず 2 ビット目以外も下げている．

Arduino は組込みシステムの学習に利用されるが，IDE やソースコード解析ツールによるミスの検知などの支援が無くミスが見逃され，潜在バグが発生しやすい．潜在バグはプログラマにとってテストなど，一般的な手法では発見が難しい [9]．スケッチ中のミスにより発生する潜在バグの除去に大きな労力を要するのは組込みシステムの学習として非効率的である．Ibrahim らは組込みシステムの教育では，コーディングのみではなくハードウェアの設計や限られた計算資源のみで動作するソフトウェアの設計を学習すべきであると述べている [10].

2.4　コードスメル

プログラミング言語一般にはコードスメルという概念がある [3]．コードスメルは，それが含まれるソースコードの保守性や可読性に問題がある兆候であり，例えば，複製されたコードや巨大なクラスなどが挙げられる．コードスメルはリファクタリングによって解消されるべきソースコードの構造であり，多くの言語でコードスメルの自動検出ツールが提案され [4]，ソースコードの品質向上が支援されている．

従来のコードスメルは特定の言語に特化した概念ではなくプログラミング言語一般の概念である．一方で，特定の言語やプラットフォームに固有のコードスメルが提案されている．Vegi と Valente はプログラミング言語 Elixir 独自のコードスメルを収集および分類し，それらが Elixir を用いる開発者にどれほど意識されているかを調査した [11] [12]．また，Wu らはコンテナ仮想化プラットフォーム Docker に用いる Dockerfile 特有のコードスメルを収集および分類し，それらが実世界の Dockerfile にどれほど存在するかなどを調査した [13]．各言語やプラットフォームに特有のコードスメルも従来のコードスメルと同様に除去することでソースコードの保守性や可読性が向上し，潜在バグの発生を低減できる．

3.　提案手法

3.1　概　　要

本研究では，Arduino の制御プログラムであるスケッチの記述時に発生する固有なミスや，ミスに起因する潜在バグの発生の低減を目指す．そこで，スケッチの記述時に発生する固有なミスをコードスメルととらえ，それらを Arduino スメルとして定義し，スケッチからの自動検出の仕組みを提案する．

3.2　Arduino スメルの定義

Arduino スメルを定義するために 2 つの方法でスケッチに固有のミスを収集する．1 つ目の方法は公式リファレンス(2) の参照である．リファレンスでは Arduino 固有の関数や定数などについて動作の概要や利用方法に加えて具体的なスケッチの例が記述されている．リファレンスには関数などの利用方法について注意が必要な場合，警告が合わせて記述されている場合がある．本研究では警告されているコードを Arduino スメルとして定義する．

2 つ目の方法は実際のスケッチの目視での調査である．明らかにプログラマの意図と異なる動作をするスケッチ特有のあるコードを Arduino スメルとして定義する．コードを収集する対象のスケッチとしてここでは，大阪大学基礎工学部情報科学科の学部 3 年生を対象に開講されている実験科目で学生が記述したスケッチ（以降，学生スケッチ）を用いる．目視による収集の対象とするスケッチには 2 つの性質が求められる．

- Arduino スメルを含んでいる
- スケッチの記述から開発者の意図が理解できる

スケッチのデータセットとして GitHub 上のスケッチや Arduino を用いたプロジェクトの共有コミュニティサイトである Arduino Project Hub(3) で公開されているオープンソースのスケッチが考えられる．しかし，これらの場所で公開されているスケッチは Arduino を用いた開発の熟練者が書いたスケッチも多いと考えられ，多数の Arduino スメルの発見は期待できない．また，オープンソースのスケッチはそれぞれ異なるシステムを実現するために記述されている．そのため，多くのオープンソース

(注2)：https://www.arduino.cc/reference/en/
(注3)：https://projecthub.arduino.cc

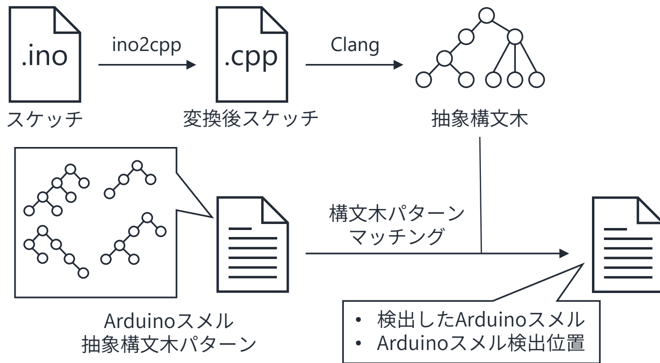

図 2　Arduino スメル検出の流れ

スケッチの意図をその記述から理解することは難しい．学生スケッチを記述した学生の多くは初学者であり，多数の Arduino スメルの発見が期待されることに加え，スケッチが与えられた同じ仕様を満たすことが確認されている．以上の理由から，ここでは学生スケッチを目視による Arduino スメルの探索対象に用いる．

3.3　Arduino スメルの検出

次に定義した Arduino スメルの検出手法を示す．提案する Arduino スメル検出手法の流れを 図 2 に示す．スケッチの抽象構文木を得て，サブツリーマッチングによって Arduino スメルを検出し，その結果をテキストとして出力する．

はじめに Clang [4]を用いてスケッチの抽象構文木を得る．しかしスケッチ（.ino ファイル）は Clang を含む C/C++ コンパイラでは解析できないため，ino2cpp [5] を用いて.cpp ファイルに変換する．.cpp ファイルへの変換はスケッチに対する Arduino 固有の関数ライブラリのインクルードとスケッチ内で定義された関数のプロトタイプ宣言，ならびに利用する Arduino ボードの宣言の追加により実現される．

次に得られたスケッチの抽象構文木を用いて Arduino スメルを検出する．検出はソースコードの静的解析手法として様々な目的に用いられている抽象構文木のパターンマッチング [14] [15] によって行う．Arduino スメルの抽象構文木パターンがスケッチの抽象構文木の部分木として存在していれば，その抽象構文木パターンに該当する Arduino スメルが検出される．ここで，マッチングに利用するための Arduino スメル抽象構文木パターンの例を 図 3 に示す．Arduino スメルの定義の拡張を容易にするために JSON 形式で抽象構文木パターンを保存する．"_msg"要素に Arduino スメルの種類を表す文字列を格納し，"_pattern"要素に対応する Arduino スメルのパターンを記述する．"_pattern"要素の第一階層に Arduino スメル抽象構文木パターンとして検出される抽象構文木の部分木の根ノードが持つ要素を記述する．あるノードが持つべき子ノードが満たす条件および子ノードの数と順序は"child"要素にオブジェクトの配列として記述する．本手法で用いる抽象構文木パターンのマッチングは完全一致以外の判定も可能である．これにより抽

(注4)：https://clang.llvm.org
(注5)：https://github.com/arquicanedo/ino2cpp

```
1  {
2    "_msg": "'millis()'の返り値が'unsigned long'以外の
         型に格納されています．",
3    "_pattern": {
4      "kind"    : "BinaryOperator",
5      "literal" : "=",
6      "child"   : [
7        {"$type": [["!", "unsigned long"]] },
8        {"exist": {"kind"    : "CallExpr",
9                   "literal": "millis"}}
10       ]
11   }
12 }
```

図 3　Arduino スメル抽象構文木パターン例

```
1  float time = millis();
2  int random_seed = SEED_SUFFIX + millis();
```

図 4　図 3 のパターンで検出可能なスメルの例

象構文木の親子関係や各ノードが持つ要素を限定しない柔軟なスメル検出を可能とする．実際に，図 4 に示された 2 行の式はいずれも関数 millis() の値を unsigned long ではない変数に代入するミスが含まれた式であり，それぞれ Arduino スメルとして検出されるべきである．しかし 2 行の式は変数の型や右辺の抽象構文木の形がそれぞれ異なり，それぞれに対して完全に一致するパターンを記述することは現実的ではない． 図 4 の例では，代入先の変数の型が unsigned long ではない，および右辺に millis() の呼び出しが存在する，の 2 つの条件により 2 行の式ともにマッチする． 図 3 では，これら 2 つの条件をそれぞれ 7 行目"$type"と 8 行目"exist"の 2 つの要素を用いて指定する．"$<要素名>"要素の記述例を 図 5 に示す．満たす

```
1  // "type" が "unsigned long" ではない
2  {"$type": [["!", "unsigned long"]]}
3  // "value" が10以上である
4  {"$value": [[">", 10]]}
5  // "value" が0でなく, 10より小さい
6  {"$value": [["!", 0], ["<", 10]]}
```

図 5　完全一致でない条件指定の例

べき条件と比較対象の値を格納したタプルの配列を渡し，否定や数値の大小比較，指定した部分文字列の有無による判定が可能である． "exist"要素にオブジェクトを渡すことで子孫ノードに条件を満たすノードが存在するかを判定できる．否定や子孫ノード中の存在によるマッチングのほかにも，3 つの完全一致でないマッチングが可能である．"parent"要素に条件を記述すれば一致するノードが先祖ノードに存在するかにより判定できる．加えて"right"要素を用いた行きがけ深さ優先探索で最後に探索するノードに対する条件の指定，および"left"要素を用いた帰りがけ深さ優先探索で最初に探索するノードに対する条件の指定も可能である．

最後に Arduino スメルの検出結果を通知する．Arduino スメルが検出されると，検出された Arduino スメルの種類と変換前のスケッチにおける Arduino スメルが検出された位置がテキストとして出力される．

4. 定義した Arduino スメル

提案手法により定義した 7 つの Arduino スメルを 表 1 に示す．以下に各 Arduino スメルがスケッチにもたらす悪影響とその選定理由を述べる．

・長すぎる delay()：10 以上の数値を与える delay() の呼び出しを Arduino スメルとして定義する．スケッチ固有の関数として，すべての計算やピンの入出力をブロックする関数 delay() がある．大きなシステムではプログラマの意図しないセンサ入力の読み飛ばしなどが発生する可能性があり，10ms 以上の delay() は公式リファレンスで非推奨とされている．

・返り値の暗黙的キャスト：関数の返り値の暗黙的なキャストは典型的な悪い兆候である．スケッチ固有の関数では特に，システムが動作を開始してから経過した時間を返す関数 millis() と micros() の返り値を，暗黙的にキャストするミスがリファレンスで警告されている．これらの関数の返り値は unsigned long 型であるため，それ以外の型の変数に代入すると，ある時間以降の値がオーバーフローしたり無視できない大きさの誤差が生じたりする．返り値の暗黙的キャストが スケッチに存在すると，システムの実行開始後オーバーフローの発生や誤差が拡大したときに初めて顕在する潜在バグが発生する．システムを実行してもすぐに返り値の暗黙的キャストが原因のバグに気が付くことができないため，提案手法のスケッチの静的解析による検出が発見に有用である．

・boolean 型の使用：変数や関数の返り値の型として boolean 型を利用する文を Arduino スメルとして定義する．かつて利用されていた boolean 型が現在も後方互換性確保のために bool 型のエイリアスとして残されている．現在は bool 型の利用が公式リファレンスで推奨されている．

・Timer/Counter0 の操作：Timer/Counter0 の設定を保持するレジスタの操作は Arduino 固有の関数の動作を意図しないものにする．Arduino 上のマイコンが持つタイマとカウンタはプログラマが利用することもでき，プログラマは対応するレジスタを操作することでタイマの設定を変更し割込みなどに利用できる．そのうち，Timer/Counter0 はスケッチ固有の関数である millis() や delay() に用いられる．Arduino は初期化時に Timer/Counter0 の設定を行っており，Timer/Counter0 を利用する関数はその設定に基づいて動作する．したがって，その設定を変更すると Timer/Counter0 を使用している関数が正常に動作しなくなる．

・割込みのネスト：割込みが発生したときに呼び出されるコールバック関数内での正しく動作しない関数の呼び出しを Arduino スメルとして定義する．スケッチでは割込み発生時に呼び出されるプロセスを関数 ISR() として定義できる．しかし，ISR() 内で，正しく動作しない Arduino 固有の関数が存在する．例えば，millis() や delay() は内部で割込み処理を用いるため，ISR() 内では正常に動作しない．

```
1 // レジスタPORTBの1,2ビットを立てる
2 PORTB |= (1 << 2) | (1 << 1);
3 // レジスタPORTBの1,2ビットを下げる
4 PORTB &= ~(1 << 2) & ~(1 << 1);
5 // L4の間違った記述
6 PORTB &= (0 << 2) & (0 << 1);
```

図 6 レジスタ操作に用いられるビット演算，およびスメルを含む演算の例

・不適切なビット演算：ビット演算のミスによりレジスタの操作をプログラマの意図と異なる方法で行う式である．スケッチ上では，Arduino ボード上のマイコンレジスタを明示的に操作する演算を記述する場合がある．レジスタ操作をするソースコード例を 図 6 に示す．スケッチにおけるレジスタ操作をするソースコードは多くの場合，いくつかの特定ビットのオンオフを切り替える意図で記述される．しかし，特定ビットを下げる意図で記述されたと考えられる演算について，0 をシフトしその値とのビット積を取る演算が学生スケッチに多く見られた．0 のシフト演算はシフトさせるビット数に関わらず 0 になるため無意味な演算である．また 0 とのビット積は特定ビットのみではなくすべてのビットを下げる演算である．不適切なビット演算に該当する演算では多くの場合レジスタのビットがプログラマの意図と異なる状態に変更されており，潜在バグの原因となる．

・不適切な割込み設定：Arduino 上のマイコンの割込み設定が適切でないとき，システムの動作がプログラマの意図と異なる場合がある．Arduino 上のマイコンにはカウンタを用いた割込み機能がある．カウンタが比較用レジスタに設定された値に到達すると割込みが発生する．割込みを無効化しないまま初期化などのために比較用レジスタを 0 に設定するソースコードが学生スケッチに多く確認された．この設定では高速で割込みが発生し，元のプロセスの実行が進まなくなる．

5. スメルの検出結果

Arduino スメル検出ツールの精度評価および Arduino スメルの定義の一般性を確認するために以下の 2 つの実験を行う．

実験 1：提案ツールが検出した集合と比較し精度を評価する．
実験 2：OSS 内の Arduino スメルの存在を確認する．

5.1 実験 1：Arduino スメル検出ツールの精度評価

5.1.1 評価手法

検出ツールの精度を評価するために自動検出の再現率（recall）および適合率（precision）を評価する．はじめに，スケッチ集合に対して目視で Arduino スメルを収集し，それらを真の Arduino スメル集合として定義する．次に同じスケッチ集合に対して提案ツールを適用し，検出された Arduino スメルの集合と真の

表 1　Arduino スメル一覧

名前	概要	出典	コード例
長すぎる `delay()`	`delay()` による 10ms 以上のブロック	公式リファレンス	`delay(1000);`
返り値の暗黙的キャスト	`millis()` と `micros()` の返り値をサイズの小さな型に格納	公式リファレンス	`int time = millis();`
`boolean` 型の使用	`bool` 型の代わりに `boolean` 型を利用	公式リファレンス	`boolean f = false;`
Timer/Counter0 の操作	Timer/Counter0 の設定をするレジスタを操作	公式リファレンス	`TCCR0A = 0;`
割込みのネスト	`ISR` 内部で割込みを行う関数を呼び出し	公式リファレンス	省略
不適切なビット演算	無意味なシフト演算と意図しないビットを変更する演算	学生スケッチ	`PORTB &= 0 << 2;`
不適切な割込み設定	高頻度な割込みにより元プロセスが実行されない割込み設定	学生スケッチ	`OCR1A = 0;`

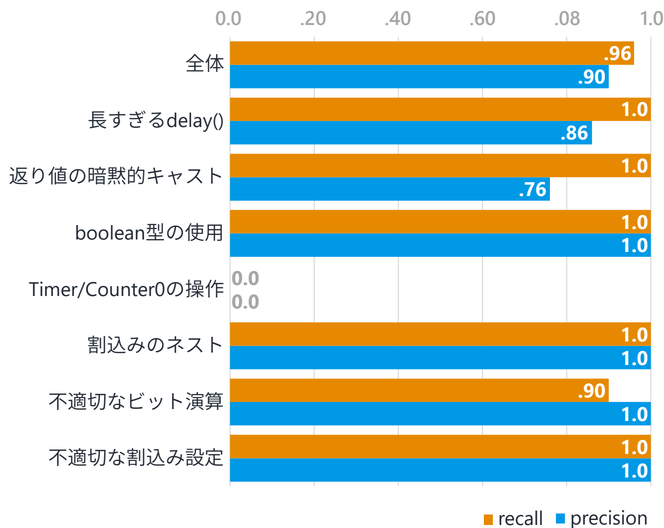

図 7　学生スケッチに対する Arduino スメルの検出精度

Arduino スメル集合を比較して提案ツールによる自動検出の再現率および適合率を評価する．

実験 1 においては学生スケッチ 221 個を用いる．真の Arduino スメル集合の定義には Arduino スメルの定義上，明らかにプログラマの意図と異なる，と判断できる記述を目視で集める必要がある．そのため，スケッチの意図が既知でなくてはならない．また，Arduino スメルを多く収集するために，対象のスケッチは熟練者が書いたスケッチより初学者が書いたスケッチが望ましい．以上の 2 点を満たすデータセットとして学生スケッチを用いる．

5.1.2　結果と考察

図 7 にそれぞれの Arduino スメルおよび Arduino スメル全体に対する提案ツールの検出精度の評価指標を示す．提案ツールは再現率 0.96 ならびに適合率 0.90 の高い精度で，目視で確認したスケッチ中の Arduino スメルを検出した．

一方で長すぎる `delay()` および返り値の暗黙的キャストの検出について，適合率がそれぞれ 0.86 および 0.76 と誤検出が比較的多く発生した．また，不適切なビット演算に該当する記述について Arduino スメルとして検出されず，見逃される記述があった．

返り値の暗黙的キャストの誤検出が多かった理由として抽象構文木パターンの定義が Arduino スメルとしてマッチすべきソースコード集合に対して過剰であったことが挙げられる．`millis()` の返り値をオーバーフローせずに受け取ることができる式として `long long` 型や `uint32_t` 型の変数で `millis()` の返り値を受け取る式を確認した．これらの式は Arduino スメルではないが，返り値の暗黙的キャストの抽象構文木パターンとマッチングしたため誤検出された．これらの誤検出は Arduino スメルの抽象構文木パターンの追加または修正により簡単に除去可能である．

長すぎる `delay()` の誤検出および不適切なビット演算に当たる記述の見逃しが発生した理由として，変数や三項演算子が含まれる記述の存在が挙げられる．長すぎる `delay()` の検出について `delay(delay_time)` のような呼び出し式が見逃されていた．このような式が Arduino スメルに該当するかを判断するには変数を動的に解析する必要がある．同様に不適切なビット演算の検出について `REG &= ((pinState == HIGH) ? 1 : 0) << 2` のような式は `inputPin` の入力が `HIGH` でないとき `REG &= 0 << 2` が実行される．不適切なビット演算を定義する抽象構文木パターンは三項演算子を考慮しないため検出されなかった．スケッチ中のすべての Arduino スメルを過不足なく検出するためには動的なスケッチの解析によって変数に渡される値やソースコードの実行経路を調べる必要がある．

また，Timer/Counter0 の操作は学生スケッチのデータセットに含まれておらず検出精度の評価はできなかった．学生スケッチが記述された実験科目では Timer/Counter0 の利用を要する課題が含まれていなかったためである．したがって Timer/Counter0 の操作の検出精度評価については別のデータセットを利用する必要がある．

5.2　実験 2：Arduino スメル一般性の確認

5.2.1　実 験 手 法

実世界の Arduino スケッチにおける Arduino スメルの存在を確認する．実験 2 においてはオープンソースのスケッチに提案ツールを適用して検出することで，どれほどの数の Arduino スメルが含まれているかの調査する．対象とするスケッチは Arduino Project Hub に公開されているスケッチ 528 個である．Arduino Project Hub 上のスケッチは実現するシステムが様々であるため，学生スケッチより一般的である．

5.2.2　結果と考察

検出したスメル数を 表 2 に示す．対象の 528 個のスケッチのうち 331 個のスケッチから定義した 7 種類のうち 4 種類の Arduino スメルの検出に成功した．4 種類の Arduino スメルはいずれも公式リファレンスで警告されている記述である．一方で

表 2 OSS スケッチに対する Arduino スメル検出数

名前	検出数
長すぎる `delay()`	1,348
返り値の暗黙的キャスト	42
`boolean` 型の使用	209
Timer/Counter0 の操作	10
割込みのネスト	0
不適切なビット演算	0
不適切な割込み設定	0

公式リファレンスに記述のなかった Arduino スメルは Arduino Project Hub 上のスケッチから検出されなかった．学生スケッチに基づく Arduino スメルの定義は，学生スケッチが記述された実験科目の題材に過度に特化している可能性がある．

また，Arduino スメルごとに検出数に大きく差があった．これは Arduino Project Hub に公開されているスケッチの多くは，プログラマによって意図したとおりの動作をすることが確かめられているためであると考えられる．検出数が多かった長すぎる `delay()` は潜在バグの原因にならない場合も多く，`boolean` 型 がスケッチに含まれていてもバグの原因にはならない．一方で返り値の暗黙的キャストに起因する潜在バグは `millis()` や `micros()` の値を代入する変数がオーバーフローすると顕在する，プログラマが比較的気が付きやすいバグである．Timer/Counter0 の操作についてもスケッチ内での `millis()` や `delay()` の利用により不具合が顕在する．気が付きやすい潜在バグの原因となる Arduino スメルは修正される可能性も高いため Arduino Project Hub 上のスケッチにほとんど現れなかったと考えられる．

実験 2 においては，完成したシステムであるオープンソースのスケッチを用いたため出現しなかった，と考えられる Arduino スメルがあった．しかし，完成したスケッチには含まれずとも記述中のスケッチに Arduino スメルが入り込む可能性がある．したがって，スケッチ記述中の Arduino スメル検出およびプログラマへの通知にはスケッチ記述の支援効果と潜在バグ発生を低減する効果が見込まれる．

6. おわりに

本研究では Arduino スケッチ特有のミスである，Arduino スメルの定義と Arduino スメルのスケッチ中からの検出を行った．結果として学生スケッチに頻出のミスや公式リファレンスの記述から 7 種類の Arduino スメルを定義し，Arduino スメルを高い精度でスケッチ中から検出可能なツールを試作した．また，試作したツールを用いてオープンソースのスケッチから定義した Arduino スメルを検出し，Arduino スメルはスケッチ一般に出現することを確認した．

今後の課題として，提案手法を用いたスケッチ記述支援ツールの開発があげられる．本研究で提案した Arduino スメル検出ツールはスタンドアローンツールであり，開発中のインタラクティブな Arduino スメル検出には利用しにくい．提案した検出手法を IDE のプラグインなどとして組み込むことでスケッチの記述と同時に Arduino スメルの検出が可能となり，Arduino を用いた開発の支援となることが見込まれる．さらに，Arduino スメルの検出がスケッチの記述支援効果を被験者実験の実施により評価する必要がある．また，スケッチの動的解析による Arduino スメル検出も今後の課題として挙げられる．提案手法ではスケッチを静的に解析したため正確な検出が難しい Arduino スメルが存在した．ハードウェアのシミュレーションを用いたスケッチ動的解析により多くの Arduino スメルの正確な検出が見込まれる．

謝辞 本研究の一部は，JSPS 科研費（JP21H04877, JP20H04166，JP21K18302，JP21K11829）による助成を受けた．

文献

[1] M. El-Abd, “A Review of Embedded Systems Education in the Arduino Age: Lessons Learned and Future Directions,” International Journal of Engineering Pedagogy, vol.7, no.2, pp.79–93, 2017.

[2] K.S. Kaswan, S.P. Singh, and S. Sagar, “Role of Arduino in real world applications,” International Journal of Scientific & Technology Research, vol.9, no.1, pp.1113–1116, 2020.

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

[4] F.A. Fontana, P. Braione, and M. Zanoni, “Automatic detection of bad smell in code: An experimental assessment,” Journal of Object Technology, vol.11, no.2, pp.1–38, 2012.

[5] 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,” IEEE Transactions on Software Engineering, vol.36, no.1, pp.20–36, 2010.

[6] A.M. Fard and A. Mesbah, “JSNose: Detecting JavaScript code smells,” In Proceedings of international working conference on Source Code Analysis and Manipulation, pp.116–125, 2013.

[7] D. Hovemeyer and W. Pugh, “Finding bugs is easy,” Journal on Special Interest Group on Programming Languages Notice, vol.39, no.12, pp.92–106, 2004.

[8] Q.L. Le, A. Raad, J. Villard, J. Berdine, D. Dreyer, and P.W. O’Hearn, “Finding real bugs in big programs with incorrectness logic,” Journal on Proceeding ACM on Programming Languages, vol.6, no.OOPSLA1, pp.81:1–81:27, 2022.

[9] Y. Brun and Michael.D. Ernst, “Finding latent code errors via machine learning over program executions,” In Proceedings of International Conference on Software Engineering, pp.480–490, 2004.

[10] I. Ibrahim, R. Ali, M. Zulkefli, and N. Elfadil, “Embedded Systems Pedagogical Issue: Teaching Approaches, Students Readiness, and Design Challenges,” American Journal of Embedded Systems and Applications, vol.3, no.1, pp.1–10, 2015.

[11] L.F.d.M. Vegi and M.T. Valente, “Code smells in Elixir: early results from a grey literature review,” In Proceedings of International Conference on Program Comprehension, pp.580–584, 2022.

[12] L.F.d.M. Vegi and M.T. Valente, “Understanding code smell in Elixir functional language,” Journal on Emprical Software Engineering, vol.28, no.4, p.102, 2023.

[13] Y. Wu, Y. Zhang, T. Wang, and H. Wang, “Characterizing the occurrence of Dockerfile smells in open-source software: An empirical study,” Journal on IEEE Access, vol.8, pp.34127–34139, 2020.

[14] I.D. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, “Clone Detection Using Abstract Syntax Trees,” In Proceedings of International Conference on Software Maintenance, pp.368–377, 1998.

[15] G. Blanc, M. Akiyama, and Y. Miyamoto, Daisukeand Kadobayashi, “Identifying Characteristic Syntactic Structures in Obfuscated Scripts by Subtree Matching,” In Proceedings of Computer Security Symposium, pp.468–473, 2011.