Arduinoプロジェクトにおける Example Sketchの再利用分析

寺村 英之
京都工芸繊維大学
h-teramura@se.is.kit.ac.jp

崔 恩瀞
京都工芸繊維大学
echoi@kit.ac.jp

近藤 将成
九州大学
kondo@ait.kyushu-u.ac.jp

吉田 則裕
名古屋大学
yoshida@ertl.jp

水野 修
京都工芸繊維大学
o-mizuno@kit.ac.jp

要旨

ソフトウェア開発を効率的に行うためのソースコードの再利用はコードクローンを作り込む原因となる．コードクローンの存在はソフトウェアの品質に影響を与えうるため，これらの調査が必要である．例えば組み込みシステムの開発に用いられる *Arduino* にはコード再利用の仕組みがあり，再利用の調査ができると便利であると考えられる．しかし我々の知る限りでは *Arduino* におけるソフトウェアプロジェクトであるスケッチの再利用に関する調査は行われていない．本研究はスケッチでの再利用調査をするために *Arduino* に特化したコードクローン検出手法を提案し，実際にスケッチから再利用と考えられるコードクローンを検出した．

1. 緒言

ソフトウェア開発では既存のソフトウェアのソースコードからその一部を再利用することがある．Sojer と Henkel がオープンソースソフトウェア開発者へのインタビューと既存の研究をもとに調査した結果，開発者は新たな機能の実装時に平均して 30%の割合でソースコードを再利用していることがわかった [1]．既存のソースコードの再利用によってコードクローンが生成される．コードクローンとは，ソースコード中に存在する互いに一致または類似したコード片を指す [2]．ソフトウェア中に存在するコードクローンを調べることでソフトウェア開発において有用な情報を得ることができる．例えばプロジェクト内でのコードクローンの調査結果に基づいて開発者は，よく使われている部分のライブラリ化や機能が似ているモジュールの統合にその情報を使うことができる．また Imoize ら [3] は再利用のメトリクスによって Reuse Library の評価を行えることを指摘している．Reuse Library とは再利用可能なソースコードを管理し，開発者によるソースコードの再利用を補助するシステムである．

Arduino はプロトタイピングや組み込みシステムの開発などに利用されている身近な電子工作プラットフォームの 1 つである．開発者が開発するソフトウェアはスケッチと呼ばれる．スケッチは Arduino ボードを PC に接続して Arduino IDE(Integrated Development Environment) から 1 クリックでアップロードするだけで動作させることができ，LED ランプなどの電子回路をその場で動作させながら開発できる．

Arduino はライブラリというソースコードの再利用を容易にする仕組みを備えている．ライブラリにはサンプルコードである Example Sketch が付属する．開発者は Example Sketch の一部を自身のプロジェクトに再利用することがあるので，コードクローンが発生する．Arduino は組み込みシステムの開発に用いられることがあり，その際は発生するコードクローンを把握することが重要である．例えば開発者がソフトウェア中のコードクローンについての情報を把握できれば，コードクローンに関係する問題が発生した場合に迅速に対応できる．また，Arduino プロジェクトからコードクローンを検出することで，ソースコードの再利用を見つけられれば，開発における Example Sketch などの役割を議論できる．

しかし，我々の調査した限りでは Arduino スケッチにおけるソースコードの再利用の調査についての研究は存在しない．

そこで本研究では Arduino におけるソフトウェア開発でのコードの再利用の調査を目的として既存のコードクローン検出ツールを組み合わせた調査方法の検討と再利用調査をした．特に Arduino における再利用の仕組みの 1 つであるライブラリの Example Sketch からのコードの再利用に注目した．この研究における研究設問を以下に示す．

RQ1 Arduino における Example Sketch からのコードの再利用を既存の方法を応用して検出するにはどうすればよいか．

RQ2 再利用されているコードの特徴はなにか．

この研究の貢献は次の 2 つである．

- Arduino スケッチにおける Example Sketch からのコードの再利用を既存のコードクローン検出ツールを応用して検出した．
- 検出されたコードクローンの内，再利用と考えられるものの特徴について議論した．

最後にこの論文の章構成について簡潔に述べる．第 2 章ではこの研究の関心の対象である Arduino とソースコードの分析に利用するコードクローン検出ツールについて説明する．第 3 章では既存のコードクローン検出ツールを応用して Arduino スケッチにおけるソースコードの再利用を検出する方法を提案する．続いて第 4 章でその手法を用いた再利用の検出結果を示す．その後第 5 章で研究設問に答えるための議論をする．最後に第 6 章で妥当性への脅威について述べ，第 7 章でこの研究の結論と貢献についてまとめる．

2. 背景

2.1. Arduino における Example Sketch

Arduino のスケッチは一般的な IDE でいうプロジェクトと対応する．スケッチには唯一のスケッチファイルと任意の数の C あるいは C++ソースコードやヘッダファイルが含まれる．作成したスケッチは Arduino IDE でコンパイルし Arduino ボードにアップロードするとボード上で動作させられる．

開発者は必要に応じてライブラリを自分のスケッチで使うことができる．ライブラリのヘッダファイルをスケッチのソースコードでインクルードするとその機能を使うことができる．

Arduino ライブラリはその使い方を説明するための Example Sketch を含むことがある．多くのライブラリでそれが提供する機能や対応しているハードウェアの種類ごとに分類されたスケッチが用意されている．これらは IDE から簡単に一覧表示でき，またボードにアップロードできる．このようなスケッチをライブラリの Example Sketch と呼ぶ．

Example Sketch はライブラリの利用時に頻繁に使うコードをしばしば含んでおりそこからの再利用はスケッチの開発においてよく行われる．例えば Ethernet ライブラリの WebClient という Example Sketch には Ethernet シールドの初期化処理と LAN への接続処理が含まれている．Ethernet シールドを使いたい開発者はこの部分をコピーして自分のスケッチで簡単にシールドを使い始めることができる．

2.2. コードクローン検出ツール CCFinderSW

本節では，コードクローンの定義とこの研究で利用したコードクローン検出ツールである CCFinderSW[4] について説明する．

コードクローンとは，ソースコード中に存在する互いに一致または類似したコード片を指す [2]．CCFinderSW は意味的ではなく，構文的に類似しているコード片をコードクローンとして検出する．例えばある配列に連続して値を保存する部分があるとする（リスト 1）．これに対してあるアドレスにあるバッファに対してデータを格納する部分を考える（リスト 2）．ここでリスト 1 の 2〜5 行目とリスト 2 の 2〜5 行目はどちらも配列への値の格納だからコードクローンである．このように完全一致するコード片だけでなく相違があるものもある種のコードクローンとして考える．コードクローンはどれだけの相違を許すかによって 4 つのタイプに分類されている [5]．

次に提案手法で利用したコードクローン検出手法である CCFinderSW について説明する．CCFinderSW はトークンベースのコードクローン検出ツールの一つである．トークンベースとはソースコードの比較にプログ

リスト 1. 配列にデータを保存するコード片

```
1 int array[4];
2 array[0] = 0;
3 array[1] = 1;
4 array[2] = 2;
5 array[3] = 3;
```

リスト 2. バッファにデータを格納するコード片

```
1 int create_packet(char * buffer) {
2   buffer[0] = 0x20;
3   buffer[1] = 0x3F;
4   buffer[2] = 0x00;
5   buffer[3] = 0xFF;
```

ラミング言語のトークンを単位として用いるという意味である．CCFinderSW がコードクローンを検出する仕組みを説明する．まず入力されたソースコードからコメントを除去し，予め与えられた文法定義に基づいて入力ソースコードをトークンに分割する．次にトークン列に含まれる英数字列を対応するプログラミング言語の予約語リストを元に予約語と識別子に分ける．その後，比較したいソースコードのトークン列同士で比較を行い一致している部分をコードクローンとして検出する．つまり識別子や数値はその値にかかわらず同じものとして扱われるのでソースコード内の変数名や定数が異なっていてもソースコードのトークン列が似ているならばコードクローンとして検出する．CCFinderSW は Java 言語で書かれたコマンドラインツールとして実装されている[1]．CCFinderSW でコードクローンを検出するには，対象のソースコードをディレクトリに集めて実行するだけで良い．また結果の出力フォーマットが決まっているので処理結果を機械的に処理することもできる．

2.3. Arduino プロジェクトでの再利用分析の必要性

本節では，Arduino プロジェクトでの Example Sketch の再利用分析の必要性について述べる．

まず，実世界にあるスケッチについてどの Example Sketch が頻繁に再利用されているのかを知ることで Example Sketch がスケッチ開発にどれほど影響を与えているか調べられる．例えば似た機能を実装している複数のスケッチに対して調査することである機能を実現することに関連したライブラリの機能を知ることができる．

[1] https://github.com/YuichiSemura/CCFinderSW

さらに，スケッチの開発者が自分のスケッチを分析することで利用していたライブラリや再利用した Example Sketch を知ることができる．これによりどのライブラリを使っていたか記録されていなかった場合にその情報を復元できる可能性がある．また，スケッチ内の Example Sketch からの再利用部分を管理することにも役立つ．

このように一般的なソフトウェアでの再利用分析の利点だけでなく，Arduino における開発特有の利点もある．

3. 提案手法

3.1. コードクローン検出

Example Sketch のコードからの再利用はコードクローンとして検出できる．例えばある Example Sketch のコードの一部がスケッチにコピーされているとする．このときその Example Sketch とスケッチとの間のコードクローンは該当再利用箇所を含む可能性がある．また，スケッチがある Example Sketch をもとにして作られている場合はその Example Sketch とのコードクローンとなる．したがって両者の間のコードクローンを検出しその結果を調べることで Example Sketch から再利用されたコードを知ることができる．

既存の研究では様々なコードクローン検出ツールが提案されている [6][7]．本研究でコードクローン検出ツールを選択する上で重要な要素は 2 つある．1 つは C++ 言語に対応していること，もう 1 つは関数単位，あるいはそれよりも細かい粒度でコードクローンを検出できることである．スケッチの主要なソースコードであるスケッチファイルとそれに付随するソースコードはすべて Arduino 言語で書かれている．そして Arduino 言語は C++言語をもとにしているので検出手法が C++言語に対応していることが重要となる．また，Example Sketch からの再利用ではファイル単位よりも小さなクローンが多く発生すると考えることができる．なぜならば開発者は Example Sketch を再利用した結果それとほとんど同じスケッチを作成する可能性が非常に低いからである．

従って，本研究では C++言語のサポートがあり，かつ細かい粒度でクローン箇所を検出できる CCFinderSW をコードクローン検出ツールとして選択した．

2.2 節で説明した通り，CCFinderSW でソースコード間のコードクローンを見つけるためには対象となるソースコードをすべて CCFinderSW の入力として与える必

要がある．したがって，スケッチと Example Sketch との間のコードクローンを見つけるにはそれらのソースコードをすべて入力すれば良い．しかし，これだけでは 1 つのスケッチに対してかなり時間がかかってしまう．実際，検出の最初の試みとして後述するスケッチ big_aquatan と様々な Example Sketch とを入力とし，研究室の計算機で CCFinderSW を実行したところ 3 日以上経っても検出が完了しなかった[2]．

この処理に時間がかかるのは入力の大きさに原因があると考えられた．まず，ライブラリの種類は IDE からデフォルト設定で検索できるものだけでも 3,560 ほどある．それぞれのリリースされたバージョンも合わせれば全体で約 16,000 種類になる．あるスケッチで使われるライブラリはこの数よりもかなり少ない上，実際に参照された Example Sketch も一部にとどまると考えられる．したがって，最新版ライブラリのすべての Example Sketch と big_aquatan のソースコードを入力として CCFinderSW に渡す場合，多くの Example Sketch に対する処理が無駄になる．また，CCFinderSW は入力として与えられたソースコードから 2 つのソースコードを選んだペアを作り，コードクローンを検出する．ところが，本研究で注目したいものは Example Sketch とスケッチの間のコードクローンである．このまま入力すれば Example Sketch 同士の分析も行われてしまうのでほとんどの処理が無駄になる．そこで効率よくスケッチと Example Sketch のコードクローンを検出を行えるように CCFinderSW への入力を作成する手法を作成した．

3.2. 作成した手法の説明

3.1 節では単に Example Sketch とスケッチを入力とすると，スケッチで使われていない Example Sketch についても分析が行われる（問題 1），Example Sketch 同士についても分析が行われる（問題 2）という 2 つの問題が生じることを説明した．

ここでは本研究で作成した手法で以上の問題をどのように解決したかを説明する．

問題 1 はスケッチと関係があると考えられる Example Sketch を絞り込むことで解決した．まず，ライブラリごとにライブラリのソースコードであるヘッダファイルの集合を作る．次に，すべての Example Sketch のスケッチファイルについてその中でインクルードされているヘッダファイルの集合を作る．その後 Example Sketch のヘッダ集合と対応するライブラリのヘッダファイル集合の積をとり，その Example Sketch の特徴ヘッダファイル集合とする．スケッチから Example Sketch を絞り込むには次のようにする．まず，スケッチのスケッチファイルでインクルードされているヘッダファイルの集合を作る．その後すべての Example Sketch の特徴ヘッダファイル集合についてそれがスケッチのヘッダファイル集合に含まれているか確かめる．そのとき，もし含まれているならば対応する Example Sketch を分析対象に入れる．特徴ヘッダファイル集合はスケッチから利用できるライブラリのヘッダファイルのうち Example Sketch で使われているヘッダファイルの集合である．そのヘッダファイルの集合がスケッチでインクルードされているということは対象の Example Sketch で使われている機能が使われている可能性が大きい．これによってスケッチで使用されようとしている機能に最も近い Example Sketch を選ぶことができる．

問題 2 はスケッチと Example Sketch のソースコードを予めペアにして入力とすることで解決した．まず，スケッチと Example Sketch のソースコードをそれぞれ 1 つずつペアにしたすべての組み合わせを作成する．次に，それらのペアごとにディレクトリを作りそれぞれのソースコードを格納する．そして作成したすべてのディレクトリを入力として CCFinderSW を実行し結果を収集する．これによって比較する意味のないペアを除いて分析できる．

3.3. 検出の条件

本節では提案手法を用いて実際に再利用検出を行った際の各種条件を述べる．

検出対象のライブラリ群として Arduino IDE のデフォルトライブラリメタデータ[3]にあるものを用いた．メタデータとライブラリのダウンロードはどちらも 2021 年 1 月 4 日から 1 月 5 日のうちに行った．ライブラリは計算リソースの都合上ダウンロードした時点で最新バージョンのものだけを検出対象にした．CCFinderSW のバージョンは 1.0 を用い，CCFinderSW のパラメータは表 1 のように設定した．

[2]CPU は Intel Xeon Silver 4210，RAM は 128GB，作業用 SSD は 512GB の NVMe SSD だった

[3]https://downloads.arduino.cc/libraries/library_index.json

表 1. CCFinderSW のパラメータ

パラメータ	値
対象の言語 (-l)	cpp
検出対象の範囲 (-w)	2（ファイル間のみ）
-antlr	”ino|pde|c|h|cpp|hpp|cxx|hxx|cc”
文字コード (-charset)	auto

この表に記述がないものは入力ディレクトリと出力ファイル名に関わるオプションである”-d”と”-o”を除いてすべてデフォルト値を指定した．また，Arduino 言語のファイルも分析対象に入れるため，grammarsv4 ディレクトリに拡張子”ino”，”pde”のための文法定義を追加した．文法ファイルは C++言語のために予め用意されている CPP14.g4 を流用した．-antlr オプションの正規表現に拡張子 hh にあたるパターンが含まれていないがこれは意図的なものでない．検出試行に用いたソースコード全てに対し拡張子が hh であるようなヘッダファイルは存在しなかったため再度試行しなくて良いと判断した．

検出試行は Example Sketch の絞り込みを行った場合と行わなかった場合の両方行った．

4. 検出の結果

検出の結果 Example Sketch を絞り込んだ場合は 5 ペア，絞り込まなかった場合は 86 ペアのソースコードからコードクローンが検出された．この結果からさらにコードが実装している機能をもとにコードクローンを分類した．それぞれの条件に分けて以下で述べる[4]．

4.1. Example Sketch を絞り込んだ場合

この条件で検出されたクローンはすべて SparkFun APDS9960 RGB and Gesture Sensor ライブラリ[5]バージョン 1.4.2 を使った初期化コードだった．特に 1 つのコードクローンはジェスチャ検出機能を有効化する部分が含まれていた．big_aquatan での該当箇所（リスト 3）を示す．

この部分は GestureTest という Example Sketch のスケッチファイル[6]の 71-88 行目および ColorSensor という Example Sketch のスケッチファイル[7]の 53-66 行目とコードクローンとして検出された．

リスト 3. スケッチの big_aquatan.ino の該当箇所

```
127 Wire1.begin(PIN_SDA1, PIN_SCL1);
128
129 eyes.begin(-5, 0, 0, 0);
130
131 if (apds.init()) {
132   Serial.println(F("LAPDS-9960␣
        initialization␣complete"));
133 } else {
134   Serial.println(F("Something␣went␣wrong␣
        during␣APDS-9960␣init!"));
135 }
136 if (apds.enableGestureSensor(true)) {
137   Serial.println(F("Gesture␣sensor␣is␣now␣
        running"));
138 } else {
139   Serial.println(F("Something␣went␣wrong␣
        during␣gesture␣sensor␣init!"));
140 }
```

SparkFun APDS9960 RGB and Gesture Sensor ライブラリは APDS-9960[8]センサの制御ライブラリである．このライブラリはセンサの初期化とジェスチャ認識などの各種機能の有効化機能を実装している．big_aquatan スケッチは APDS-9960 センサを用いてハンドジェスチャを認識する機能がある． また，該当部分のメッセージ文字列やコードの書式がよく似ており，検出された部分は再利用された部分であると言える．

これらの部分は 5 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 5 個だった．

4.2. Example Sketch を絞り込まなかった場合

この条件で検出されたクローンは次のように分類できた．

分類 1 SparkFun APDS9960 RGB and Gesture Sensor ライブラリを使った初期化コード

[4] この結果の再現用のデータなどは次の場所にある．: https://github.com/ikubaku/ss2021-scripts

[5] https://github.com/sparkfun/APDS-9960_RGB_and_Gesture_Sensor/

[6] https://github.com/sparkfun/APDS-9960_RGB_and_Gesture_Sensor/blob/V_H1.0_L1.4.2/Libraries/Arduino/examples/GestureTest/GestureTest.ino

[7] https://github.com/sparkfun/APDS-9960_RGB_and_Gesture_Sensor/blob/V_H1.0_L1.4.2/Libraries/Arduino/examples/ColorSensor/ColorSensor.ino

[8] https://www.broadcom.com/products/optical-sensors/integrated-ambient-light-and-proximity-sensors/apds-9960

リスト 4. スケッチの big_aquatan.ino の該当箇所

```
653 DateTime now = rtc.now();
654 char str[20];
655 sprintf(str, "%04d-%02d-%02d␣%02d:%02d:%02d"
        , now.year(), now.month(),
656         now.day(), now.hour(), now.minute(),
                now.second());
657 ts = str;
658 return ts;
```

分類 2 リアルタイムクロックの時刻取得コード

分類 3 SNTP 関連のコード

分類 4 arduino-esp32 の OTA イベント処理コード

分類 5 再利用ではないコード

4.2.1. 分類 1 に属するコードクローン

分類 1 に属するコードクローンの数は 5 つだった．これは Example Sketch を絞り込んだ場合に検出されたものだと考えられる．なぜならこの条件で比較した Example Sketch の中に絞り込んだ場合の Example Sketch が全て含まれている上，検出されたクローンが同じ 5 箇所だったからである．

4.2.2. 分類 2 に属するコードクローン

分類 2 に属するコードクローンは主にリアルタイムクロックから取得した時刻を書式化する処理を実装していた．big_aquatan での該当箇所（リスト 4）は MCP7940 ライブラリの SetAndCalibrate という Example Sketch のスケッチファイル[9]の 173-180 行目とコードクローンとして検出された．

コードクローンを含む Example Sketch に対応するライブラリはすべてリアルタイムクロックの制御ライブラリだった．これらの Example Sketch も多少の改変を除けば現在時刻を取得し文字列に初期化する処理を含んでいた．しかし，big_aquatan では sprintf 関数の行の改行位置が now.day メソッドの呼び出し直前にある．それに対し Example Sketch では now.year や now.hour メソッド呼び出しの直前で改行されていた．また書式文字列は ISO8601 に基づいた日時表現とよく似ているほか，現在時刻のアクセス部分も対応するメソッドを呼び出すだけで実現できるので特殊なものであるとは言えない．したがってこの部分が再利用であるか判断できなかった．

これらの部分は 8 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 10 個だった．

4.2.3. 分類 3 に属するコードクローン

分類 3 に属するコードクローンはさらに 2 種類に分類できた．1 つは SNTP パケットを作成するコード，もう 1 つは SNTP サーバから受信したデータをもとに 1900 年からの経過秒数を計算するコードだった．big_aquatan での該当箇所（リスト 5，6）はそれぞれ RTCDue ライブラリの RTCDue_NTP_WIFI_with_timezone という Example Sketch のスケッチファイル[10]の 183-203 行目，log4Esp ライブラリの AdvancedDemo という Example Sketch のスケッチファイル[11]の 137-143 行目とコードクローンとして検出された．

SNTP パケットを作成する部分は識別子や周辺のコメントなどが一致しており，再利用されたコードであると言える．

一方で受信したデータを処理するコードはスケッチのものと Example Sketch のものの間で違いが見られた．big_aquatan のコードでは Example Sketch のコードで実装されているタイムゾーンに合わせて秒数を調整するコードが失われている．また packetBuffer[43] の直後の 0 ビットシフトは Example Sketch のコードには存在しなかった．それでもスケッチと Example Sketch のコード片には識別子などの共通点が見られるので，共通の源流となるコードを持つなど何らかの関係があると考えられる．

これらの部分は 55 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 171 個だった．このことからコードクローンとして検出された部分は様々なライブラリの Example Sketch で再利用されていることがわかる．

[9] https://github.com/Zanduino/MCP7940/blob/v1.2.0/examples/SetAndCalibrate/SetAndCalibrate.ino

[10] https://github.com/MarkusLange/RTCDue/blob/RTCDue_Version_1.1/examples/RTCDue_NTP_WIFI_with_timezone/RTCDue_NTP_WIFI_with_timezone.ino

[11] https://github.com/hunsalz/log4Esp/blob/v1.0.1/examples/AdvancedDemo/AdvancedDemo.ino

リスト 5. スケッチの ntp.cpp の SNTP パケット作成部分

```
10
11 // send an NTP request to the time server at
       the given address
12 void NTP::sendPacket() {
13
14   // set all bytes in the buffer to 0
15   memset(packetBuffer, 0, NTP_PACKET_SIZE);
16   // Initialize values needed to form NTP
       request
17   // (see URL above for details on the
       packets)
18   packetBuffer[0] = 0b11100011; // LI,
       Version, Mode
19   packetBuffer[1] = 0; // Stratum, or type
       of clock
20   packetBuffer[2] = 6; // Polling Interval
21   packetBuffer[3] = 0xEC; // Peer Clock
       Precision
22   // 8 bytes of zero for Root Delay & Root
       Dispersion
23   packetBuffer[12] = 49;
24   packetBuffer[13] = 0x4E;
25   packetBuffer[14] = 49;
26   packetBuffer[15] = 52;
27   // all NTP fields have been given values,
       now
28   // you can send a packet requesting a
       timestamp:
29   udp.beginPacket(timeserver.c_str(), 123);
       //NTP requests are to port 123
30   udp.write(packetBuffer, NTP_PACKET_SIZE);
```

リスト 6. スケッチの ntp.cpp の受信データ処理部分

```
37 unsigned long secsSince1900 = 0;
38 // convert four bytes starting at location
       40 to a long integer
39 secsSince1900 |= (unsigned long)packetBuffer
       [40] << 24;
40 secsSince1900 |= (unsigned long)packetBuffer
       [41] << 16;
41 secsSince1900 |= (unsigned long)packetBuffer
       [42] << 8;
42 secsSince1900 |= (unsigned long)packetBuffer
       [43] << 0;
43 return secsSince1900 - 2208988800UL; //
       seconds since 1970
```

4.2.4. 分類 4 に属するコードクローン

分類 4 に属する検出部分には OTA のエラーイベント処理の実装が含まれていた．big_aquatan での該当箇所（リスト 7）は IRremoteESP8266 ライブラリの IRrecvDumpV3 という Example Sketch の BaseOTA.h[12] の 39-54 行目とコードクローンとして検出された．

リスト 7. スケッチの big_aquatan.ino の該当箇所

```
162 .onProgress([](unsigned int progress,
        unsigned int total) {
163   digitalWrite(2, (progress / (total / 100))
        % 2);
164   Serial.printf("Progress:␣%u%%\r", (
        progress / (total / 100)));
165 })
166 .onError([](ota_error_t error) {
167   Serial.printf("Error[%u]:␣", error);
168   if (error == OTA_AUTH_ERROR)
169     Serial.println("Auth␣Failed");
170   else if (error == OTA_BEGIN_ERROR)
171     Serial.println("Begin␣Failed");
172   else if (error == OTA_CONNECT_ERROR)
173     Serial.println("Connect␣Failed");
174   else if (error == OTA_RECEIVE_ERROR)
175     Serial.println("Receive␣Failed");
176   else if (error == OTA_END_ERROR)
177     Serial.println("End␣Failed");
178 });
179 ArduinoOTA.begin();
180
181 webServer.on("/", handleStatus);
182 webServer.on("/reboot", handleReboot);
```

OTA とはネットワーク上にあるデバイスのソフトウェアのアップデートを可能にする機能である．Arduino の場合は ArduinoOTA ライブラリでその機能を実現している．検出された部分では”ArduinoOTA”という識別子が出現するのでこの機能に関係があると考えられる．しかし，ArduinoOTA ライブラリ[13]の Example Sketch と検出部分を比較したところ，検出された部分は ArduinoOTA には存在しなかった．更に調査したところ検出された部分と似た部分を以下の Example Sketch が見つかった．

- arduino-esp32 の BasicOTA[14]
- ESP8266/Arduino の BasicOTA[15]

これらはそれぞれ ESP32 シリーズと ESP8266 マイクロコントローラ向けの Arduino ソフトウェアの実装に含まれる Example Sketch である．実際，検出されたコー

[12] https://github.com/crankyoldgit/IRremoteESP8266/blob/v2.7.14/examples/IRrecvDumpV3/BaseOTA.h
[13] https://github.com/jandrassy/ArduinoOTA
[14] https://github.com/espressif/arduino-esp32/blob/master/libraries/ArduinoOTA/examples/BasicOTA/BasicOTA.ino
[15] https://github.com/esp8266/Arduino/blob/master/libraries/ArduinoOTA/examples/BasicOTA/BasicOTA.ino

リスト 8. スケッチの actionqueue.h の該当箇所

```
34 public:
35   actionQueue(AquatanEyes *e, AquatanArms *a
         , Camera *h,
36               NeoPixels *c); //, Charger *c);
37   void queueEyes(int l_shape, int r_shape,
         int m);
38   void queueArms(int left, int right, int
         time);
39   void queueArmLeft(int left, int time);
40   void queueArmRight(int right, int time);
41   void queueHead(int pan, int tilt);
42   void queueHeadPan(int pan);
43   void queueHeadTilt(int tilt);
```

ドクローンを含む Example Sketch に対応するライブラリはすべて ESP32 か ESP8266 を用いたシステムを前提としていた．したがってこれらの Example Sketch でもコードが再利用されたと考えられる．

コードクローンを観察するとエラーメッセージやエラーコードを確認する際の比較の順番などが一致しており，検出された部分は再利用されている可能性があると考えられる．特に big_aquatan の該当部分は ESP8266/Arduino の BasicOTA にある部分とよく似ていた．したがって該当部分はその Example Sketch の実装を源流としている可能性がある．

これらの部分は 6 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 30 個だった．

4.2.5. 分類 5 に属するコードクローン

分類 5 に属するコードクローンは再利用ではないと考えられるもので，さらに 2 種類に分類できた．

- 関数プロトタイプ宣言とクラスメソッド宣言
- 配列への値の保存

検出例をそれぞれリスト 8，9 とリスト 10 に示す．Example Sketch の配列への値の保存部分はすべてスケッチの SNTP パケット作成部分（リスト 5）とペアになっていた．

big_aquatan のソースコードに含まれる関数プロトタイプ宣言やクラスメソッド宣言は Example Sketch の名前

[16] https://github.com/hideakitai/ES920

[17] https://github.com/madhephaestus/Esp32SimplePacketComs

リスト 9. ES920[16]-0.1.9 の openFrameworks/ascii/src/ofApp.h の該当箇所

```
13 void keyReleased(int key);
14 void mouseMoved(int x, int y );
15 void mouseDragged(int x, int y, int button);
16 void mousePressed(int x, int y, int button);
17 void mouseReleased(int x, int y, int button)
       ;
18 void mouseEntered(int x, int y);
19 void mouseExited(int x, int y);
20 void windowResized(int w, int h);
21 void dragEvent(ofDragInfo dragInfo);
22 void gotMessage(ofMessage msg);
```

リスト 10. Esp32SimplePacketComs[17]-0.7.0 の SwarmRobotExample/SwarmRobotExample.ino の該当箇所

```
13 // User data is written into the buffer to
       send it back
14 void event(float * buffer) {
15   int numberOfFloats = 15;
16   buffer[0] = 10; // X Location
17   buffer[1] = 0; // Y Location
18   buffer[2] = 0; // Z Location
19   buffer[3] = 45; // azimuth
20   buffer[4] = 0; // elevation
21   buffer[5] = 0; // tilt
22   buffer[6] = 100; // X size
23   buffer[7] = 100; // Y size
24   buffer[8] = 100; // Z size
25 // Serial.println(
```

にある異なる関数とペアとして検出された．big_aquatan スケッチにあったこれらの関数プロトタイプ宣言やクラスメソッド宣言はすべて独自の機能に関係するものだと考えられるので再利用ではないと言える．

また，ntp.cpp に含まれる SNTP パケット作成処理の部分と Example Sketch で見られる様々な配列への代入部分もコードクローンとして検出されていた．これらは処理の意味が異なっており再利用ではないと考えられる．

これらの部分は 12 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 39 個だった．

5. 議論

この章では前の章の結果に基づいて議論し，研究設問に回答する．

5.1. 再利用の検出手法

まず研究設問 1 に回答する．

再利用の検出試行では提案手法を用いて再利用と考えられるいくつかのコード片を発見できた．Example Sketch の絞り込みのありなしにかかわらず比較は 3 日以内に完了し，再利用ではない検出結果も少なかった．したがってこの手法は現実的なリソースで Example Sketch からの再利用を探すことに一定の効果があると言える．

一方で提案した Example Sketch の絞り込みが再利用されたコードの発見を妨げていることもわかった．検出できなかったコードは第 4 章の分類 2 にあたるもの，分類 3 にあたるもののうちパケット送信を行うコード，分類 4 にあたるコードの 3 つである．それぞれについて検出できなかった原因として考えられるものを議論する．

まず分類 2 のコードについて考える．Example Sketch を絞り込んだときリアルタイムクロック関係のものとして RTClib の Example Sketch が対象に含まれていた．しかし RTClib の Example Sketch には発見されたような書式化処理はなかった．つまりヘッダファイルの共通性から Example Sketch を絞り込むと似たような機能を持つ異なるライブラリからの再利用を検出できないと考えられる．

次に SNTP パケット送信コードがなぜ検出できなかったについて議論する．このコードは UDP 通信機能を使っているあるいは提供しているライブラリの Example Sketch にあった．例えば log4Esp の AdvancedDemo では WifiUdp.h が include されており，WiFi ライブラリの UDP 通信機能が SNTP 通信のために使われている．一方 big_aquatan ではスケッチファイルでなく ntp.h の中で include されていたため絞り込みの際にそのヘッダを考慮に入れることができていなかった．したがってどのヘッダが include されているのかを絞り込みの情報に使うことはそのままでは難しく，ヘッダファイル内の include プリプロセッサ命令まで考慮する必要があるとわかった．

分類 4 のコードが検出できなかったことには他の要因が関わっている．ライブラリをダウンロードする際は，ライブラリメタデータに登録されているものをダウンロードした．しかし，これらには Arduino コアが提供するライブラリは含まれていない．Arduino コアとは Arduino ボードやマイクロコントローラの種類ごとに実装されている中核となるソフトウェアである．これにはボードに書き込まれるブートローダや入出力など基本的な操作を提供するコード，そしていくつかのライブラリが含まれている．コアに含まれるライブラリは基本的な機能を提供する少数のライブラリとボード特有の機能を提供するライブラリがある．arduino-esp32 の ArduinoOTA ライブラリや WiFi ライブラリもこのライブラリの一種である．これらのライブラリの Example Sketch も考慮に入れたい場合は Arduino コアのライブラリからも Example Sketch を予め集めておく必要があると言える．

5.2. 再利用の特徴

次に研究設問 2 に回答するため，前の章の発見できたコード片の分析をまとめる．

ここでは再利用であるか判断できなかったコードを省いた，分類 1，分類 3 のうちパケット送信を行うもの，分類 4 に当たるコードそれぞれについて特徴をまとめる．

分類 1 や分類 4 のコードはエラーの提示方法を工夫するなどの動機がない限り改変する必要のないコードだった．また，SNTP パケット送信コードについても時刻を取得するだけならば内容を工夫する必要はない．実際 Blynk ライブラリ[18]ではこの部分が BlynkSimpleEthernetSSL.h に取り込まれている．したがって再利用されているコードにはある機能を実装するコードスニペットあるいは一般的なソフトウェア開発でのライブラリとも言える特徴があると考えられる．

しかしこの検出試行で発見できた再利用の種類は 3 種類にとどまった上，対象としたスケッチも 1 つのみであるため一般的な Arduino における開発に一般化できない．再利用の特徴についてはさらなる調査が必要であると言える．

6. 妥当性への脅威

検出試行では計算リソースの都合上ダウンロードした時点で最新のライブラリのみ検出に使用した．したがって，もし古いバージョンにのみ見られるコード片が big_aquatan で再利用されていた場合はその部分を検出できない．このライブラリバージョンの制約は妥当性への脅威となりうる．

また，提案した手法を実現するツールは執筆者が開発し，必要と思われる部分にはテストを行った．しかし，

[18] https://github.com/blynkkk/blynk-library

ツールにまだ未知のバグが含まれている可能性は否定できず，妥当性への脅威となりうる．

7. 結論

本研究では Arduino プロジェクトにおける Example Sketch からのコードの再利用を CCFinderSW への入力を工夫して作成することで分析し，再利用されている部分の特徴を調べた．一方で，再利用部分を見つけただけではそのコードがどこから再利用されたのか明らかにできないという課題が見つかった．

本研究における今後の課題は以下のとおりである．まず，再利用元や再利用の目的を明らかにするため，開発者にインタビュー調査を行う予定がある．また，提案手法において入力データの絞り込み方法に課題があった．今後は提案手法を課題を明確化し手法の改良や結果の応用方法について研究を進めるべきであると考える．

参考文献

[1] M. Sojer and J. Henkel, “Code reuse in open source software development: Quantitative evidence, drivers, and impediments,” Journal of the Association for Information Systems, vol.11, no.12, pp.868–901, 2010.

[2] 肥後芳樹，楠本真二，井上克郎，“コードクローン検出とその関連技術,” 電子情報通信学会論文誌. D，vol.91，no.6，pp.1465–1481，jun 2008.

[3] A.L. Imoize, D. Idowu, and T. Bolaji, “A brief overview of software reuse and metrics in software engineering,” World Scientific News, vol.122, pp.56–70, 2019.

[4] 瀬村雄一，吉田則裕，崔恩瀞，井上克郎，“多様なプログラミング言語に対応可能なコードクローン検出ツール CCFinderSW,” 電子情報通信学会論文誌 D，vol.103，no.4，pp.215–227，2020.

[5] C.K. Roy, J.R. Cordy, and R. Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,” Science of computer programming, vol.74, no.7, pp.470–495, 2009.

[6] H. Sajnani, V. Saini, J. Svajlenko, C.K. Roy, and C.V. Lopes, “SourcererCC: Scaling code clone detection to big-code,” In Proc. of ICSE, pp.1157–1168, 2016.

[7] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, “Deckard: Scalable and accurate tree-based detection of code clones,” In Proc. of ICSE, pp.96–105, IEEE, 2007.