# プログラム理解のためのコードの機能分けツールの検討

西本 匡志[1,a)]　西山 佳志[2]　川端 英之[1]　弘中 哲夫[1]

**概要**：ソフトウェア開発では様々な場面で他者が実装したプログラムを理解することは重要である．しかし，行数が多く構造が複雑なコードから実装された機能やその処理の詳細を把握することは容易ではない．そこで我々はコードで使用される API の関連に着目して機能に分け，個々の機能とコードの対応箇所を把握できるツールを提案する．このツールでコードに含まれる機能を可視化し，コード理解の支援を目指す．

## 1. はじめに

複数の開発者が関わるアプリケーション開発では，レビューや不具合修正，あるいは機能追加を行うために，他者が実装したコードを理解することは重要である．しかし，大規模，あるいは構造が複雑なコードでは実装された機能の抽出やその内容の理解は容易ではない．

コードの理解を行うにあたっては，トップダウンとボトムアップの 2 つの視点がある．トップダウンの視点ではコードから UML のクラス図やシーケンス図を自動生成するツールを用いて，プログラムの処理の流れや構造を把握することができる．ボトムアップの場合には CASE ツールによる型情報や変数宣言箇所の一覧表示，デバッガによるステップ実行でコードの詳細内容を把握可能である．しかし，これらのツールは使用方法の理解や専門知識の習得が必要であり，使用への障壁は低くはない．

これに対し，本発表ではトップダウンにコード理解を行う際に特別な技能や専門知識を必要とせず，対象のコードにおける API の関連性から機能分けを行い，その機能毎に処理の概要を表示するツールを提案する．機能毎の処理の表示には，直感的な内容把握の支援に向くとされるタグクラウドを用いる（タグクラウドは API の使用法の理解 [1] や利用状況の可視化 [2] 等で使われている）．また，本ツールは個々の機能とコードの対応箇所を提示する機能を備えている．本ツールは，トップダウンとボトムアップの両視点からコードの俯瞰を可能にするものと期待される．

## 2. コードの機能分けツールの基本方針

開発者がコードを読む際，そのコードに実装された機能やその処理について把握しようと試みる．例えば，コードの字面に表れる語句から機能や処理を推測し，エディタの検索機能を用いて制御やデータの依存関係を把握しながら，推測した機能や処理に関係のあるコードの箇所を探す．この作業を何度も繰り返すことにより，開発者はコードの全体像，あるいは個々の機能やその詳細なフローなどを理解していく．しかし，この方法では巨大なコードを読み解く際に労力を要する上に，不正確なコードの理解を行ってしまう可能性がある．

これに対し，我々は，次のような機能を持つツールが効率的にコード理解を支援できるのではないかと考えている．

(1) ユーザがコードを与えれば，そのコードを構成する機能を列挙できる．
(2) 列挙された個々の機能の概要を端的に提示できる．
(3) 列挙された各機能に関連する箇所を明記できる．

これらを踏まえたツールを実現するにあたり，我々は，多くのプログラムがライブラリの API を組み合わせて構築されていることに着目する．そして，コード中に現れる API を相互の関連性に基づいて分類することにより，コードに対する機能ごとの階層的分割の実現を目指す．

API 同士の関連性は，変数を介して形作られるデータ依存関係に基づいて判断することができる．ここで，図 1 のコードを例に，我々の手法の考え方について述べる．図 1 では，変数 *mWindowManager* で指されるオブジェクトの生成に API “getSystemService” および名前付き定数 “WINDOW_SERVICE” が使用されている．また，変数 *rotation* で指されるオブジェクトは，オブジェクト *mWindowManager* および API “getDefaultDisplay” によって生成され，また，*rotation* の値はオブジェクト *mDisplay* と API “getRotation” によって決定されている．この様子は図 2 左のデータフローグラフによって示される (図中，四角および丸はそれぞれ API 名および変数名を表している)．

1 広島市立大学大学院 情報科学研究科
2 広島市立大学 情報科学部
a) nishimoto.masashi@ca.info.hiroshima-cu.ac.jp

```
54 : public class Sample extends Activity {
         ...
58 :     private PowerManager mPowerManager;
59 :     private WindowManager mWindowManager;
60 :     private WakeLock mWakeLock;
61 :     private Display mDisplay;
         ...
65 :     @Override
66 :     public void onCreate(Bundle savedInstanceState) {
             ...
73 :         mPowerManager = (PowerManager) getSystemService(POWER_SERVICE);
74 :         mWindowManager = (WindowManager) getSystemService(WINDOW_SERVICE);
75 :         mWakeLock = mPowerManager.newWakeLock(
76 :                 PowerManager.SCREEN_BRIGHT_WAKE_LOCK, getClass().getName());
77 :         mDisplay = mWindowManager.getDefaultDisplay();
             ...
86 :     }
87 :     @Override
88 :     protected void onResume() {
             ...
95 :         mWakeLock.acquire();
96 :         int rotation = mDisplay.getRotation();
97 :         switch (rotation) {
                 ...
114:         }
             ...
120:     }
121:     @Override
122:     protected void onPause() {
             ...
129:         mWakeLock.release();
             ...
135:     }
         ...
430: }
```

**図 1**　Java による Android アプリケーション記述例

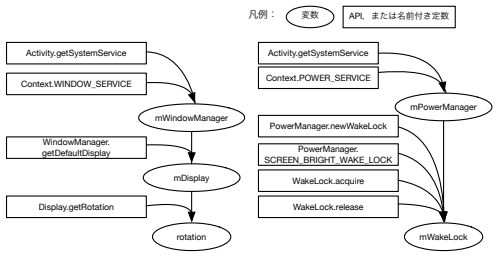


**図 2**　図 1 のコードにおけるデータフローグラフ

図 2 左のグラフから変数 *rotation* の値の生成を通して，4 つの API が相互に関連していることが分かる．同様に，図 1 のコードからは図 2 右のデータフローグラフで表される関係も抽出でき，6 つの API が互いに関連していることが把握できる．

データ依存関係に基づいて得られる API の集合は，それぞれが特定の機能の実装に関連しているとみなせる．そして，個々の API の集合がどのような機能に関連しているかは，API 名や API の所属クラス名を構成する単語から類推できるだろう．また，単語の出現頻度を鑑みれば，個々の機能を代表する単語を抽出することもできそうである．こうして得られる（重み付きの）単語の集合と API の集合を関係付けて，API の集合ごとの各 API の出現箇所の提示機能と連携させることにより，(1)-(3) の機能を持つツールの実現が期待できる．

## 3.　プロトタイプの実装と実行例

我々は 2 節で述べた基本方針を踏まえ，ユーザが Android アプリケーションのコードを入力すれば，機能一覧（図 3 の左画面）と，個々の機能に関するコードの箇所（図 3 の右画面）を出力するツールのプロトタイプを実装した．

本プロトタイプは次の 2 つの処理から構成されている．

(1) データ依存関係に基づいて入力コードから API 集合を抽出する．

(2) (1) で得た API 集合から単語を抽出し，その出現頻度からタグクラウドを生成する．単語は API 集合の要素の名前（API や所属クラス）からキャメルケースなどの命名規則を踏まえて抽出する．

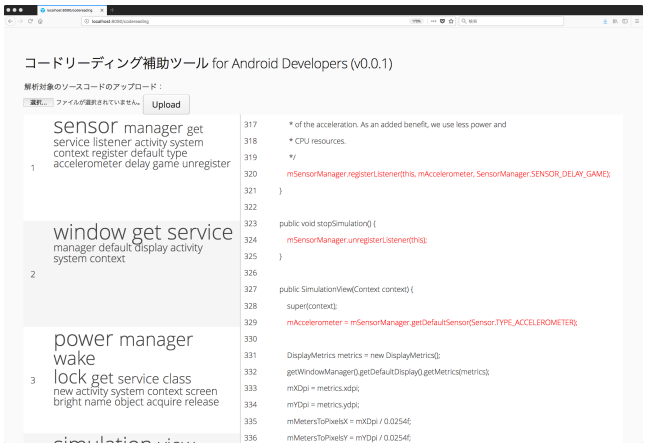


**図 3**　ツールのプロトタイプの外観と実行例

図 3 の実行例を踏まえて，ツールの効果について考察する．図 3 の機能一覧の上から順に流し読みすれば，「センサーの管理や取得」，「ウインドウの取得」，「電源管理の起動」に関する機能だと直感的に確認できる．このような機能の把握の容易さは，適度な機能分割およびタグクラウドの利用の効果の表れである．図 3 はユーザが「センサーの管理や取得」機能を選んだ場面を示しており，コード中の “getDefaultSensor”，“registerListener”，“unregisterListener” の 3 つの API が強調表示されている．この様子を見れば，ユーザはこれらの API が当該機能に対応したものだと把握できる．このように，本ツールを使えば，開発者のコード理解に対する大幅な負担軽減が期待できる．

本ツールは次のような用途も期待できる．(1) ユーザが注目したい API が記述された行を選択すれば，その API を含む機能が提示され，ボトムアップの視点からコード理解を支援できる．(2) コード全体が見えるように縮小すれば，ある機能の実装箇所を分布図として表示でき，メソッドやクラス化などのリファクタリングを支援できる．

## 4.　おわりに

本発表では，プログラム理解の支援を目指し，コードに実装された機能を可視化するツールを提案した．今後はツールの有効性に対する客観的指標の検討が必要である．

### 参考文献

[1] Bajracharya, S., Ossher, J. and Lopes, C.: Searching API usage examples in code repositories with sourcerer API search, *Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, ACM, pp. 5–8 (2010).

[2] De Roover, C., Lammel, R. and Pek, E.: Multi-dimensional exploration of api usage, *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, IEEE, pp. 152–161 (2013).