# 集約の観点から見たソースコード類似性の判定

肥　後　芳　樹†

本稿では，2 つのコードが集約可能であるかという視点からソースコード類似性の判定を行う．素直で単純な方法で集約できるソースコードは類似性が高く，集約に高次な工夫を必要とするコードは類似性が低いと著者は考える．

## 1. はじめに

著者はこれまでの数年間，コードクローンに関する研究（検出手法，分析・可視化手法，集約手法，修正支援手法など）を行ってきている．本稿では，集約（コードクローンを 1 つにまとめる）という視点から，類似コードの扱いについて記述する．なお，著者は Java 言語については約 8 年間の経験があり，研究に関わるソフトウェアを複数（合計行数で約 10 万行）開発してきた．

著者なりの類似コードの（あいまいな）定義を示しておく．ソースコードのテキスト表記や，その他の表現方法，例えば，制御フローグラフや，プログラム依存グラフが類似（または完全に一致）しているコードを類似コードという．

## 2. アンケート回答

アンケートの回答を表 1 に示す．

**観点 A**　2 つのコード片は類似しているので，1 つのコードにまとめたほうがよい．本稿では，この操作を「集約」とよぶ．

**観点 B**　2 つのコード片は類似しているので，一方にバグがあれば，他方にも同様のバグがあるか調べたほうがよい．

**観点 C**　2 つのコード片は類似しているので，一方のコードを再利用できそうな場所では，かわりに他方を使ってもよい．ただし，1 節に示した著者の類似コードの定義を用いた場合は，類似コードではないが再利用可能な場合があることに注意されたい．それは，たとえば，ソース No.8 のようにソート結果は等しくなるが，実装アルゴリズムが異なる場合である．

**観点 D**　2 つのコード片は類似しているので，類似しているという事実を記録して管理したほうがよい．

**表 1**　設問への回答

| ソース No. | A | B | C | D | X |
|---|---|---|---|---|---|
| 1 | yes | yes | yes | no | - |
| 2 | yes | yes | yes | no | - |
| 3 | yes | yes | yes | no | - |
| 4 | yes | yes | no | no | - |
| 5 | yes | yes | yes | no | - |
| 6 | no | no | no | no | - |
| 7 | yes | yes | yes | no | - |
| 8 | no | no | yes | no | - |
| 9 | no | yes | no | no | - |
| 10 | yes | yes | yes | no | - |
| 11 | no | no | no | no | - |

各ソースへの回答は，下記の条件が満たされているという前提のもとで行った．

- ライセンスに関する問題，例えば，ライセンスが異なるので 1 つのコードにまとめることができない，や再利用できない，などはない．
- 各ソースのペアは同じソフトウェア内に存在するコードである．

なお，著者は，コード片レベルの類似情報を記録・管理することについては，その有効性に疑問をもっているため，全ての回答について no としている．

### 2.1 ソース No.1 についての回答理由

クラス名が異なるのみであり，ロジックは全く同じである．これらが別々に存在している必要はない．

### 2.2 ソース No.2 についての回答理由

アクセス修飾子が異なるのみであり，ロジックは全く同じである．アクセス修飾子が public のコードのみあればよいと思われる．

† 大阪大学
Osaka University

### 2.3 ソース No.3 についての回答理由

インターフェース名が同じであり，宣言されているメソッドも 1 つを除き同じであるので，これら 2 つのインターフェースが独立して存在している必要はないのではないか．*getElementByID* が定義されているインターフェースのみを残すか，もしくは，これら 2 つのインターフェースに階層関係を持たせることによって，重複をなくすことがよいのではないだろうか．

### 2.4 ソース No.4 についての回答理由

これらのクラスは明かに用途が異なる（IPv4 と IPv6）．そのため，これら 2 つのクラスの機能は共に必要である．しかし重複部分が目立つようであれば，*Extract SuperClass* パターンや *Form Template Method* パターンを使って重複部分をなくすのがよいだろう．

### 2.5 ソース No.5 についての回答理由

用いているリスト型が異なるのみである．しかし用いているリスト型に依存した処理は特にはなく，どちらか 1 つが存在していれば良いだろう．

### 2.6 ソース No.6 についての回答理由

ジェネリクスで型を指定しているか指定していないかの違いである．型の完全性に重点をおくのであれば，ジェネリクスで型を指定した使用が望ましいだろう．インターフェース名も同じであるし，まとめたほうがよいのではないか．以下にまとめた際のコードを例示する．

```
interface AccessibleStateSet<T> {
    public Vector<T> states = null;
}
```

### 2.7 ソース No.7 についての回答理由

メソッドの一部が抽出されているのみで，ロジックは全く同じである．

### 2.8 ソース No.8 についての回答理由

「ソートを行う」という機能は同じであるが，その実現方法が異なる．著者の立場では，これは類似コードではない．

### 2.9 ソース No.9 についての回答理由

それぞれ，8 バイトの数値と 16 バイトの数値を扱うクラスである．しいて問題点をあげるとすれば，共通ロジック部分に同様のバグが潜んでいる可能性がある．

### 2.10 ソース No.10 についての回答理由

メソッドの一部が抽出されているのみで，ロジックは全く同じである．

### 2.11 ソース No.11 についての回答理由

変数名は同じであるが，型がことなるため，ロジックが異なる．重複コードとよべるのか疑問．

## 3. 議　　論

集約を目的とした場合，類似しているか否かの判断は，集約可能であるか，もしくは集約するべきであるか，になるだろう．類似コードを集約する方法は，単に類似部分を新しいメソッドとして抽出する簡単な方法や，分断して存在している共通するロジック部分を処理の流れとして抜き出して共通化する方法など，さまざまなやり方が考えられる．筆者は，より素直で簡単に集約可能なコード片は類似度が高く，集約に工夫を必要とするコード片は類似度が低いと考える．また，集約を目的とした場合は（ほかの場合もそうかもしれないが），人によってコード片の類似度は異なると考える．つまり，2 つのコード片が与えられたとき，開発者がその 2 つのコード片を集約する良い方法を思いつけば，それらのコード片は類似しており，思いつかない場合は，それらのコード片は類似していないとなる．

人による類似度の違いをなくすためには，ツールにより，与えられた 2 つのコード片を集約する「最善」の方法を提示することが必要になるが，自動的に「最善」の方法を導き出すのは不可能ではないかと思う．なぜなら，ソースコードを解析結果からそのようなモデルを作ること自体が非常に難しいだろうし，また，「最善」の集約方法は，これまでの修正の履歴や今後の保守の予測なども必要になると考えられるからである．

ワークショップでは，他の参加者に，集約を目的とした場合のコード片の類似度に関すること，例えば以下のようなことについて意見をいただければと思う．

- 類似度を定義するために考慮すべき要素．
- 類似度を算出することの可否とその理由．
- 類似コードの集約として有効な支援方法．