ソースコード類似性:理解支援の立場から

鈴　木　正　人†

コードの類似性の判定は字面の共通性によって行われる場合が多いが，本来の目的はコードの利用者がその役割を理解し，統合や再利用が可能かを判断する指針を与えることにある．理解支援の立場から類似性判定には字面だけでなく意味情報を考慮すべき理由について述べる．

1. はじめに

筆者は 1992 年に言語処理系の応用に関する研究で学位を取得後，一貫してソフトウェア開発環境に関する教育・研究に従事している．Java 言語は 1996 年の言語仕様公開時からツールの開発および研究の素材として使用し，また C/C++コンパイラや Java VM などの言語処理系の原理や構成方法を講義している．

最近ではソフトウェアの理解支援に興味を持っており，大量のソースの中から機能追加やバグフィックスの際に変更すべき箇所を抽出する手法を研究している．本稿では理解支援の観点からソースコードの類似性について議論する．

2. アンケート回答

コード理解はまず各プログラム単位 (Java ではクラス) の「役割」を把握することからはじまる． そこで観点 X は「クラスの役割が同一と解釈可能か」とした．役割は開発者がプログラム全体の中でどのような働きをさせるかを表したものである．多くの場合開発者の意図を反映し，また UML 図などより上位の設計成果物とも密接に関係している．

2.1 ソース No.1 についての回答理由

クラス名のみの相違であり，メソッド等も同じであるため同一と解釈できる．ただしクラス名が具体的な意味を含んでいる場合には，(たまたまプロパティ，メソッドが全て同一だとしても) 将来別々の拡張を行うことを見越して別のクラスとして定義されているという可能性があるので，ケース毎に検討するべきである．

† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

表 1　設問への回答

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

2.2 ソース No.2 についての回答理由

メソッドのスコープをサブクラスに限定するかどうかの相違は，クラスの機能や使われ方に影響するため，まとめるべきではない．ただし設計段階で十分考慮されていた場合に限られる．

2.3 ソース No.3 についての回答理由

特定のメソッドの公開/非公開は，No.2 と同様の理由で機能に影響するため，まとめるべきではない．

2.4 ソース No.4 についての回答理由

メソッドの機能 (IPv6 のサポートの有無) が異なるのでまとめるべきでない．4-1 で`preferIPv6Address` が false の際に`hostName` の値がどうなるか不明だが，もし 4-2 に相当する機能が実現されているならば上位互換性を持つと解釈すべきである．

2.5 ソース No.5 についての回答理由

5-2 はおそらくは効率向上を意図して GNU クラスを採用したものと推測される．機能は同一だが実装に開発者の意図が含まれており，まとめることは適切でない．ただしアルゴリズムは同じなので，片方のバグがもう片方にも残っている可能性はある．

2.6 ソース No.6 についての回答理由

Generics の判断は難しい． Java5 に不慣れな開発者が (意図的でなく) うっかり忘れた可能性が否定できないからである． もし 6-1 が`Vector<Object>`と書かかれていれば，開発者が意図的に型を非限定にしていることが明確である．いずれにしてもコードの信頼性が異なるのでまとめることはできない．

2.7 ソース No.7 についての回答理由

Fowler の「Extract Method」操作であり，明らかに意図をもって行われている (ただし効果は不明)．リファクタリング操作の前後での類似性検証の際には，操作が正しく行われたかを検査する必要がある．また「Move Method」操作も行われた場合には，クラス単位の比較では類似性を検証できない可能性もある．

2.8 ソース No.8 についての回答理由

アルゴリズムが異なるものはまとめるべきではない．類似物として管理すべきかは，管理目的が仕様中心か実装中心かに依存する．一般にはコードの字面が異なるものは別物とするべきか．

2.9 ソース No.9 についての回答理由

値の有効範囲の違いは明らかに意図的な変更である．目的も異なるはずでまとめることは考えにくい．ただしこれがラッパークラスの定義でなく，単に複数のクラス間でプロパティの型のみが異なっている場合は，それが意図的かどうか個別に判断する必要がある．

2.10 ソース No.10 についての回答理由

これは関係が理解しにくい．10-1 から 10-2 が生成されたと仮定するならば,「Extract Method」の例と考えられるが，逆方向の変換はまずありえない．いずれにしてもまとめることは不適切であり，また結果が読みにくい上バグも共通に存在している可能性が高い (最新の Tomcat6 API では`StoreAppender` クラスは [storeconfig パッケージ全体が] 廃止になっている)．

2.11 ソース No.11 についての回答理由

クラス名の相違については No.1 と同様，プロパティの型の相違はその意図により機能や信頼性に影響する．

ただし 11-1 はプロパティの初期値が 0(一般には偽), 11-2 は初期値が`true`(真) となっているので機能が異なる可能性がある．しかしこの断片だけでは 11-1 を使用するクラスで 0 が本当に偽を表す値として扱われているかは判断できない．

3. 議　　論

「C:再利用可能かどうか」という観点は目的が理解しにくい．仕様に書かれた概念や実装 (アルゴリズムを含む) に共通性があって初めて再利用の是非が議論できるのであり，単に結果のコード A,B を比べて「A が再利用できるなら，B も再利用できるか」を判断することはできない．このためほとんどの例で no と回答している．

「D:管理すべきか」に関しても，共通部分を持つコード片の組は類似物とするのが自然であり，共通部分が少ない No.8 以外は何らかの形で類似あるいは派生として管理すべきである．これも上位互換性など，共通性に関する厳密な定義を与えないまま，管理情報を利用することは難しい．

「X:役割が同じか」はプログラムの利用者からみた類似性のひとつの視点を表している．内部構造と機能が完全に同じならば，実装が異なっても役割は同じである．例えば No.5 ではライブラリ，No.8 ではアルゴリズムが異なるが，クラスの機能は共通である．ただし No.3 は interface なので公開メソッドの違いを役割の違いとして判定したが，もしこれが class であったなら，公開メソッドが部分集合の関係にあるクラスの役割は同じ (あるいは 3-1 は 3-2 の拡張) と解釈すべき場合もある．

内部データ (プロパティ型) とロジックの異なる No.11 の判断は難しい．もし 11-2 が`i=false` であったなら型の変更のみと解釈され，役割は共通である可能性が高い． 論点を明確にするためには，型だけを変更した例とロジックだけ変更した例を別々に用意すべきであろう (No.9 はラッパークラス定義なので純粋な型の変更とはみなせない)．

コード理解では役割と共にクラス間の関係 (アーキテクチャ) を把握することが重要であり[1)]，類似性はそのために必要な情報のひとつと考えられる．類似性判定に意味情報を含める場合，意味情報を言語モデルの要素として記述する方法がある．例えば 11-1 と 11-2 の相違点は字面的には (クラス名，プロパティ型，プロパティ初期値) であるが，初期値を変更した理由を「プロパティの役割」として記録する．

現在のところは，スライスなどの機械的に解析できる情報を除けば，理解に必要な情報は解析結果を利用者に示し，それに対するフィードバックとして得ているが，言語モデルの構成要素毎に類似/相違の情報を詳細に分析・提示することで，類似性の判定および理解支援に有用な情報が獲得できると考えられる．

参 考 文 献

1) Diomidis Spienellis, トップスタジオ訳, ”Code Reading:オープンソースから学ぶソフトウェア開発技法”, 毎日コミュニケーションズ, 2004.