局所的なソースコード類似性の判断は，文脈や意図に依存するので難しい

権　藤　克　彦†

1. はじめに：著者の立場

- アンケート回答者のソースコード類似性とのかかわり：ソースコード類似性の研究はしていない．ソースコード解析ツールなどの研究や，主に C 言語で開発する立場から，ソースコード類似性に興味を持っている．
  経験としては，15 年以上前に，某ネット掲示板 (C 言語で約 10 万行) の移植作業を行い，端末処理の類似コードが遍在していて驚いた経験がある．
- アンケート回答者のソフトウェア開発経験年数：20 年．ただし，一年のうち一ヶ月しかコードは書いていない．
- Java に関わってきた経験年数：10 年．
- Java との関わり方：授業で教えている，Java 1.2 の教科書を書いた，最近は読まないし書かない．Java モニタが Mesa スタイルなことは知ってるが，ジェネリクスの文法はよく知らない（パラメタ型多相だとは知ってる）．

2. アンケート回答

- 観点 A について：「まとめることを検討すべき」なら yes が増えるが,「まとめる」と断定的になると no が増える．後者の立場で回答．
- 観点 B について：安全側に倒して,「似てるのであれば念のため確認したほうが良いだろう」ということで，yes になりがち．
- 観点 C について：「かわりに他方を使ってよい」の意味で微妙に答えが変化する．「代替物として（そのまま）使ってよい」なのか「代替物として使えるように修正することを前提として使ってよい」なのか．ここでは後者で回答．
- 観点 D について：観点 B と同じで，安全側に倒して，yes になりがち．

表 1　設問への回答

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

2.1 ソース No.1 についての回答理由

コードも短く，たまたま似ているだけと判断．

2.2 ソース No.2 についての回答理由

アクセス修飾子の違いが「慎重に検討した末」なのか「なんとなく」なのかで答えが変わる．ここでは前者として回答．

2.3 ソース No.3 についての回答理由

これだけ共通部分があるとまとめたくなる．

2.4 ソース No.4 についての回答理由

インタフェースが異なるのだから，無理にまとめる必要は無い．プロトコル独立な API も存在するが，疑わしき一般化 (speculative generality) の可能性が高いと思う．

2.5 ソース No.5 についての回答理由

No.2 と類似して,「慎重に検討した末に List と TIntArrayList を使い分けているのか」なのか「うっかり」なのかで答えが変わる．ここでは後者で回答．

† 東京工業大学，Tokyo Institute of Technology

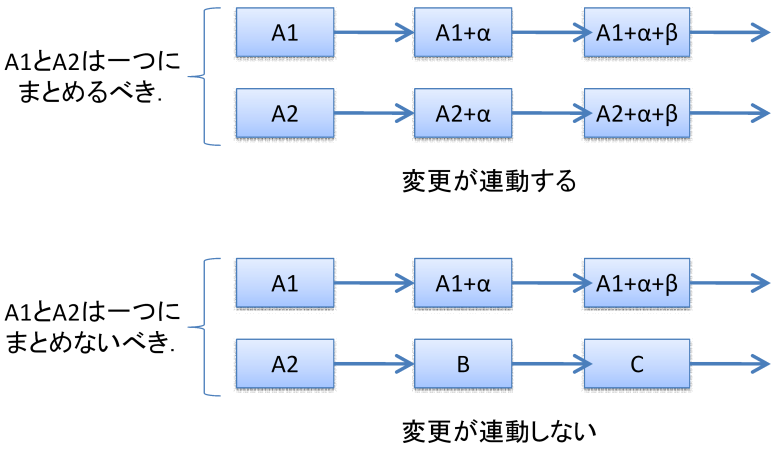

図 1 連動する変化と連動しない変化（A1 と A2 は類似コード）

2.6 ソース No.6 についての回答理由

普通はジェネリクスを使うと判断するのかも知れないが，変数 `states` に何を入れたいのか，どう使いたいのかに依存して，ジェネリクスを使わないのもありだと判断した．

ただ，この 2 つが完成した同じソースコード中にあれば「統一すべき」と思う．

2.7 ソース No.7 についての回答理由

`private` でメソッド抽出する意図によるが，あまり意図を感じないので，この 2 つが完成した同じソースコード中にあれば統一すべきと判断．

2.8 ソース No.8 についての回答理由

「インタフェースが同じで異なる実装を提供する」という意図があるという仮定で回答．

2.9 ソース No.9 についての回答理由

No.4 に類似して，無理にまとめる必要は無い．まとめたいけど，無理にまとめると理解しにくくなるパターンと判断．

C 言語であればマクロの使用を検討するが，悩んだ結果「まとめない」という決断を下しそう．

2.10 ソース No.10 についての回答理由

No.7 と同じ理由．

2.11 ソース No.11 についての回答理由

No.1 と類似して，コードも短く，たまたま似ているだけと判断．この先，コードがどう変化する（はず）なのかによる．

3. 議　　論

類似したコードをまとめるかどうかは，将来にわたって，**そのコードの変更が連動するか否か**が大きな要因になる (図 1)．変更が連動しないのであれば，その 2 つのコードはまとめない方が良いから．(後で分割するハメになって痛い目にあったことがあります．)

将来にわたるコードの変更は「未来を予測・予言すること」なので，局所的なコードの類似性だけでは判断がつかない．文脈，意図，プロジェクトの方針や状況などに大きく左右されるはず．例えば，納期が厳しくてリファクタリングする時間がないケースを想定すると，自明なモジュール化だけをするという選択になるはず．