類似コードをまとめるためのコード記述時における開発者の利用

林　　晋平[†]

類似しているソースコード片を 1 つにまとめることは有用だが，そのためにはまとめる効果とコストのトレードオフを考慮した判断が求められる．効果や変更可能性の判断の一部を該当コードの開発者によりコード記述時に行うことが，コード片をまとめる判断のコスト低下につながる．

1. はじめに

筆者は現在，リファクタリングをはじめとするソースコード変更の分析及び支援手法について大学で研究している．リファクタリングのなかには類似したコード片の共通部分を抽出するものがあり，これを行うことによりコードの保守性の向上が期待できる．次節でのアンケートにおける観点 A「2 つのコード片をまとめる」コード変更は，筆者らの研究における支援対象のひとつである．

著者は，12 年強のソフトウェア開発経験を持っている．筆者は情報工学科，計算工学専攻といったソフトウェアに関わりのある学科・専攻を卒業・修了しており，この期間は 9 年弱である．報酬を伴うソフトウェア開発経験はパートタイムのもののみで，企業でのフルタイムでの開発経験はない．12 年のうち，Java は約 7 年利用している．特に，ソフトウェア開発支援のためのツール構築に Java を用いることが多い．例えば，統合開発環境 Eclipse のプラグイン開発を行うことがあり，その実装は Java で行う．また，そのなかでは Java ソースコードの解析も行う．

本稿では，以上のような立場から，ワークショップ参加者のためのアンケートへの回答を示すとともに，ソースコード類似性に関する意見を述べる．

2. アンケート回答

ワークショップ Web サイト[1)] に示されたアンケートへの回答を行った．表 1 に結果を示す．各項目の意味は Web サイトを参照されたい．

各観点での類似性を考える上では，与えられた 2 つのコード片が同一プロジェクトに属しているか否かを考えながら判断を行った．例えば，観点 A を判断する際には，ひとつにまとめることを検討していることから同一のプロジェクトに属していると想定し，2 つのコード片のクラス名が等しい場合は別のパッケージ下にあるものと考えた．一方，観点 B を考える上では再利用の対象を外部のプロジェクトであると想定し，特にプロジェクトの同一性は考えなかった．

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	no	yes	no	-
4	yes	yes	yes	no	-
5	no	yes	yes	yes	-
6	yes	yes	yes	no	-
7	yes	yes	yes	no	-
8	no	no	yes	yes	-
9	no	yes	yes	yes	-
10	yes	yes	yes	no	-
11	no	yes	yes	yes	-

回答を行った結果，観点間に関係がみられた．例えば，観点 D は ¬ 観点 A ∧ (観点 B ∨ 観点 C) と一致した．これは，記録のコストを想像し，コード片を共通化すべきであれば記録は不要であると考え，かつ，記録の利用法としてバグの調査や再利用を考えたためである．

以下にいくつかのコード片に対する選定理由や雑感ををを述べる．

2.2 ソース No.2 についての回答理由

No. 2 で与えられた 2 つのコード片はメソッド `changeIndent` の可視性が異なる．観点 B について，両コード片にではフィールドへの代入のみが振る舞いとして記述されており，バグが生じる余地はないと考

† 東京工業大学
Tokyo Institute of Technology

えるが，ここでは実際にバグが生じた場合を検討して yes とした．

2.3 ソース No.3 についての回答理由

No. 3 の両コード片はインタフェースであり，実装を含んでいないためバグの調査は不要であるとした．これは，API 仕様に問題があった場合でも，特定のクラスを実装する際に問題が起こるため，インタフェース間の重複に意識的である必要はないと考えたためである．

2.4 ソース No.4 についての回答理由

No. 4 では共通のスーパークラスを導入することにより重複の排除が行える可能性がある．もっとも，引き上げ可能な振る舞いやフィールドは多くない．

2.5 ソース No.5 についての回答理由

No. 5 では，利用しているリストの実装が異なっている．`gnu.trove.TIntArrayList` は `java.util.List` を実装（implement）しないリストの一実現であるため，多くのメソッドシグネチャは`ArrayList<Integer>` と一致するものの，Java の機構では使用コードの共通化が難しい．

2.8 ソース No.8 についての回答理由

No. 8 は，ソートアルゴリズムの 2 実装である．観点 C について，再利用の目的が特定のソートアルゴリズムなのかソート一般かで可能性が異なるが，ここでは前者を考え no とした．

2.9 ソース No.9 についての回答理由

No. 9 では多くの字句が共通であるが，Java の機構からは重複の共通化が難しい．

2.11 ソース No.11 についての回答理由

No. 11 のコード群は No. 9 のコードのように型に関する違いが表現されており，Java ではこれらを共通化することが難しい．

3. 議　　論

実際にソースコードの類似性を認識し，記録や共通化などの保守のアクティビティを行うためには，そのコストと効果のトレードオフを考え，効率よく判断を行う必要がある．例えば，観点 A の「まとめたほうがよい」は，まとめて不都合があるか否かの判断か，まとめる手間を含めての総合的な判断であるかによって結果は異なり得る．本稿では（設問が類似コード片の規模，量的特徴を問題にしているわけではないことを鑑み）前者の観点から回答を行ったが，実際の開発では後者の判断も求められる．前者の判断には，コードをまとめても該当プログラムの外的振る舞いが変わらないこと等の確認に必要なコードの理解が求められる．また，後者の判断は，コード片をまとめる修正や，他の開発者への修正の公知にかかる労力に見合うコードの規模かの判断から，対象モジュールが今後修正される価値があるかの判断をも含む．これらの判断は簡単ではないため，判断の結果まとめられるコード片の規模が小さい場合は割に合わない．

本稿では，判断のコストを下げて効果的にコードをまとめることに強く貢献可能な開発者は，該当コード記述時の記述者であると主張する．前述の判断コストを削減し，類似コード片をまとめるなどによりコードの保守性を効果的に高めるためには，コードの理解を効率よく行う必要がある．また，対象ソフトウェアが開発の途中であれば，最終的な規模の判断には見積もりが求められる．コード記述者は，これらに必要な，対象コードの修正意図を持っている．

コード記述時の記述者から，コードの類似性の判断やそれに必要な情報を抽出することは有用だろう．例えば，コード修正時に，その修正の様子からコードをまとめるか否かの判断を促すこと[2] はそのひとつである．また，実際にコードをまとめずとも，記述対象のコード片に類似したコード片の候補を特定し，外的振る舞いに関する確認を行うだけでも貢献は大きい．もちろん，これらの判断は開発者のコード記述を強く妨げることなく行われる必要があるため，そのユーザインタフェースや判断のコストは注意深く設計されるべきである．機能追加等のコード記述時に，コードの保守性を意識し，その品質向上を同時に試みることは広く行われている[3]．ソフトウェア開発において，コードの類似性を意識し，記録や共通化などのアクティビティを行うことが重要なのであれば，それが効率よく行われるよう，保守に限らず多様な開発工程においてその実施が検討されるべきと考える．

参 考 文 献

1) 石尾, 山本: ソースコードの類似性ワークショップ, `http://sel.ist.osaka-u.ac.jp/JWSCS2009/`.
2) Hayashi, S., Saeki, M. and Kurihara, M.: Supporting Refactoring Activities Using Histories of Program Modification, *IEICE Trans.*, Vol. E89-D, No. 4, pp. 1403–1412 (2006).
3) Murphy-Hill, E., Parnin, C. and Black, A. P.: How We Refactor, and How We Know It, *Proc. 31st International Conference on Software Engineering*, pp. 287–297 (2009).