情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-SE-166 No.20
2009/11/6

ソースコードの類似性ワークショップ開催報告

石尾　隆[†1]　山本哲男[†2]　佐々木裕介[†1]

ソフトウェアエンジニアリングシンポジウム 2009 併設ワークショップ「ソースコードの類似性」の実施概要および成果について述べる．

A Report on the Japanese Workshop on Source Code Similarity 2009

Takashi Ishio,[†1] Tetsuo Yamamoto[†2] and Yusuke Sasaki[†1]

This paper reports the result of the Japanese Workshop on Source Code Similarity co-located with Software Engineering Symposium 2009.

1. はじめに

ソフトウェア工学分野において，類似したソースコードを検出するための技術や，それを利用したソフトウェア開発支援手法が数多く研究されている．たとえば，コードクローン検出手法[3] やそれを利用したリファクタリング支援[7]，開発者が編集中のソースコードに似たコード片を見本として提示する手法[8] などが挙げられる．しかし，種々の研究において，発見するべき類似したソースコードの定義はそれぞれ異なっている．Coady らは，多数のコードクローン検出技法について，それらの手法の特徴と違いを整理している[4]．また，定義が異なる複数のツールから得られたコードクローン情報の差分を計算することで，望む種類のコードクローンだけを抽出して利用する手法も提案されている[6]．

これまでに提案されている類似ソースコード検出ツールは，それぞれが定義した「類似」の基準を満たすソースコードをすべて検出する．そのため，検索手法の評価において使われる再現率，適合率という評価指標のうち，適合率は常に 100 %となり，複数の手法のトレードオフなどを比較評価することが困難である．Bellon ら[1] は，対象ソフトウェアに意図的に複製したソースコードを埋め込み，また，コードクローン検出ツールの出力の一部を手作業で検査することで，探索対象となるコードクローン集合を決定し，6 つのコードクローン検索技術を比較している．この方法は，ツールごとの検出の差異を調べるために効果的であったと考えられるが，どのようなソースコードが検出されるべきであるか，という点については明示されていない．

本ワークショップは，ソフトウェア工学の中でもプログラム解析技術の研究者を中心に，類似したソースコードとはどういうものかを明らかにすることを目的として開催した．互いに類似していると思われるソースコードの候補を著者らが事前に列挙しておき，それらが似ているか否かを参加者が手動で分類していく共同作業を実施した．

本稿は，ワークショップで得られた成果物を分析し，その結果を共有することを目的とする．原稿にすべてのデータを掲載することはできないため，使用したソースコードや参加者が記載した回答用紙などの詳細なデータについては，ワークショップウェブサイト[10] に掲載しているものを参照していただきたい．以降，2 節ではワークショップにおける共同作業の実施手順を説明し，3 節にてその成果物の分析結果を解説する．4 節には妥当性への脅威についての考察を，5 節には，今後，同様の共同作業型ワークショップを計画する際に役立つと思われる情報を整理した．最後に，6 節にまとめを述べる．

2. 共同作業の実施手順

2.1 アンケートに基づく参加者のグループ化

参加者には，ワークショップのポジションペーパーとして事前アンケートへの回答を投稿してもらった．このアンケートは 11 組のソースコードを例示しており，それらをリファクタリングすべきか，一方でバグが発見されたときに他方もまた調査すべきか，といった 4 つの観点と，著者の研究独自の観点からソースコードが類似していると考えるかどうかを，yes/no によって回答する方式となっている．

アンケートの結果を参考に，この時点で判明していた参加予定者 16 名を 4 名ずつ 4 グループに分けた．グループ参加者が議論で発言しやすくなるように，以下の基準を設けた

†1 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†2 立命館大学 情報理工学部
College of Information Science and Engineering, Ritsumeikan University

事前のグループ分けを行っておき，会場で当日参加者 2 名および遅刻者に対する微調整を行った．

- 同一所属の参加者は，可能な限り別グループに入れる．
- 学生が所属するグループには，アンケート回答の類似度が近い教員を最低 1 人入れる．
- アンケート回答の類似度が低い者同士は同じグループに入れない．

なお，アンケート回答の類似度には，単純なハミング距離を用いた．参加者 2 名の間で，ソースコード 11 組それぞれについて 4 つの観点で答えられた yes/no の回答が一致した割合を計算した．アンケート回答は，平均で約 75 %が一致しており，最も類似度が低かった参加者の組で 4 割の一致，最も類似度が高かった組で 9 割の一致となっていた．

参加者のソフトウェア開発経験には個人差が大きい．学生の参加者は，研究等のための開発経験が 2 年から 3 年程度であり，教員の参加者は，10 年から 20 年程度であると回答している．また，参加者全員が C 言語または Java を使用したことがあるが，Java の開発経験を明示的に回答しているのは 9 名であった．Java のソースコードの意味を理解することに関しては問題は少ないが，Java の細かい言語仕様については理解に差があると考えられる．参加者の多くはプログラム解析技術を研究しており，研究対象として Java を解析している者，つまり言語仕様の多くを知っているであろう参加者は各グループに最低 1 名，最大 3 名含まれている．

2.2 題材となるソースコードの準備

本ワークショップで使用したソースコードは，ソースコード部品グループ化ツール Twigi[9] とコードクローン検出ツール CCFinderX[2] を用いて抽出した．第 3 著者がツールを適用してソースコード 59 組を選び出し，その中から第 1 著者が以下の基準にしたがってソースコード 36 組を選出した．

(1) 同一の要因で類似していると判定された組は代表 1 組だけを残す．たとえば A と B，A と C，B と C がそれぞれクラス名だけの違いにより類似していると判定されたとき，代表 1 組だけを残す．

(2) ソースコードを印刷するため，ソースコードが短いものを優先する．

Twigi は，引数として，グループ化する部品に要求する類似度（0 から 1 の範囲の実数値）の閾値 x を取り，与えられた Java クラスの集合を「類似した」クラスのグループへと分解する．2 つのクラス C_1, C_2 が類似していると判定されれば，それら 2 つのクラスを同一のグループに所属させる．あるグループに所属するクラス C からみると，グループ内には少なくとも 1 つ，C と類似していると判定されるクラスが存在するが，グループ内部のすべてのクラスと互いに類似しているとは限らない．

Twigi は，2 つのクラス C_1, C_2 について以下のすべての条件が成り立つとき，2 つのクラスが互いに類似していると判定する．なお，各条件において使用されるクラス名については，そのパッケージ名を無視して，クラス名が同一であれば同じクラスを指しているものとして扱う．また，条件の定義に登場する集合間の類似度 sim と数値間の類似度 D の算出方法については後述する．

- C_1, C_2 のいずれも，最も外側で定義されたクラスである（内部クラスではない）．
- C_1, C_2 のいずれも，同名のクラスを継承している．
- C_1, C_2 の名前が，単語列を `CamelCase` 形式（数字，$やアンダースコアなどの記号も単語区切りと考える）で連結したものと仮定したとき，C_1 と C_2 に少なくとも 1 つの共通の単語が含まれている．
- クラス C で宣言された private ではないメソッドのシグネチャ集合を $Methods(C)$ としたとき，$sim(Methods(C_1), Methods(C_2)) \geq x$ である．
- クラス C で宣言された private ではないフィールドのシグネチャ集合を $Fields(C)$ としたとき，$sim(Fields(C_1), Fields(C_2)) \geq x$ である．
- クラス C で宣言された private ではないコンストラクタのシグネチャ集合を $Cons(C)$ としたとき，$sim(Cons(C_1), Cons(C_2)) \geq x$ である．なお，コンストラクタは戻り値と名前を持たないので，シグネチャ情報とは引数列の情報のみである．
- クラス C が呼び出すメソッドとコンストラクタの集合（コンパイル時の静的な呼び出し先）を $Calls(C)$ としたとき，$sim(Calls(C_1), Calls(C_2)) \geq x$ である．
- クラス C の定義の開始から終了までに出現するすべての型名の集合から C 自身を除いたものを $Types(C)$ としたとき，$sim(Types(C_1), Types(C_2)) \geq x$ である．
- クラス C のソースファイルに出現する全トークン数を $len(C)$ としたとき，$D(len(C_1), len(C_2)) \geq x$ である．ここでのトークンとは，ソースコードの字句解析の単位のことで，空白文字によって区切られた文字列である．
- クラス C のソースファイルに出現する全トークンの種類数 $kind(C)$ に対して，$D(kind(C_1), kind(C_2)) \geq x$ である．トークンの種類とは，抽象構文木におけるノードの種類である（たとえば「識別子」は 1 種類のノードである）．

上記の条件で使用されている集合の類似度 sim は，集合 S の要素数を $|S|$ と表記すると，次の式で表現できる．

Vol.2009-SE-166 No.20
2009/11/6

表 1 ワークショップのプログラム.

予定時刻	実際の時刻	内容
10:00-10:10	10:00-10:10	オープニング
10:10-12:00	10:10-11:30	自己紹介およびポジションペーパー紹介（5 分/人）
	11:30-12:00	共同作業内容の事前説明
13:00-16:20	13:00-17:00	共同作業
16:30-17:30	17:05-17:30	議論およびクロージング

表 2 グループのソースコード閲覧順序.

セッション	A	B	C	D	備考
1	P	Q	R	S	グループ A のうち 1 名は参加できなかった.
2	Q	R	S	P	
3	S	P	Q	R	
4	R	S	P	Q	ここまで時間内に閲覧した.
5	T	U	U	T	実際は閲覧しなかった.
6	U	T	T	U	

$$sim(S_1, S_2) = \min(\frac{|S_1 \cap S_2|}{|S_1|}, \frac{|S_1 \cap S_2|}{|S_2|})$$

また，トークン数および種類数の類似度 D は，次のように求める．

$$D(v_1, v_2) = \frac{\min(v_1, v_2)}{\max(v_1, v_2)}$$

本ワークショップにおけるソースコードの選択では，オープンソースソフトウェアの集合に対して様々な閾値で Twigi を適用し，類似度の閾値を $x < 1$ でグループ化されるが $x = 1$ ではグループ化されない（完全には一致しない）クラスの組を抽出した．また，CCFinderX[2] による部品の抽出では，同様のソースコード集合に対してクローン検出を適用し，他のソースファイルとの間でコードクローンになっているコードの割合（RSA）が 0.8 以上のものを抽出した．

2.3 ソースコード閲覧の手順

本ワークショップで予定していたプログラムと，実際の時間を**表 1** に示す．共同作業セッションが長引いたことから，予定よりも議論時間が短くなった．共同作業では，参加者が適宜休憩を取れるよう，全体の作業時間を 30 分から 45 分程度の「セッション」に分解し，1 セッションに一群のソースコードを評価していくという方式を採用した．ソースコード 1 組に対する判断に必要な時間を 5 分と見積もり，用意した 36 組のソースコードをランダムに 6 組ずつに分解し，P, Q, R, S, T, U とした．各グループへのソースコードの割当てを**表 2** に示す．この割当ては，閲覧の順序が結果に影響しないよう，どの 2 つのソースコード群の組についても，2 グループごとに閲覧順序が入れ替わるようにしている．参加者のグループ分けには人為的な判断が加わっているが，グループ A, B, C, D という記号の割当て自体はランダムに行った．

各グループには，6 組のソースコードを 1 部ずつ，その差分を対照できるよう着色して印刷した紙を配布した．ソースコードの差分の印刷には，WinMerge[5] というソフトウェアを用いた．また，参加者が持参したノート PC 上でキーワード検索等を行えるよう，ソースコードの組 (`.java` ファイル) と `diff` コマンドの出力結果，印刷した紙と同一の PDF ファイル，WinMerge のバイナリを会場にて配布した．

各グループは，与えられた 6 組のソースコードを自由な順序で読解し，以下の手順で類似性の判定を行った．

- ソースコードが類似しているかどうか，1 人ずつ個人での判定結果を回答用紙に記述する．
- グループのメンバー全員の判定結果を相互に確認し，判断の理由を述べ，議論を行う．
- 再度，各個人の判定結果を回答用紙に記述する．

この手順により，各参加者の最初の意見と，グループでの討論によって注目された要素を記録することを目指した．

2.4 作業コンテキストの設定

参加者は，ソフトウェア部品の再利用を支援するために，ソースコード検索エンジンを構築する立場に立っているものとした．構築するシステムは，いわゆるキーワード検索により，対応するソースコードをランキング表示すると仮定する．このようなシステムが提示するべきソースコードをデータベースに蓄積するためにインターネット上からソースコードを収集してきたとき，互いに類似したソースコードが存在する可能性がある．

参加者は，与えられた組のコードが実際に収集された場合にどうするべきかを，以下の中から選択して回答する．

(a) 一方のみを残す． 一方だけを登録し，他方は登録しない．

(b) 集約する． 2 つのソースコードを 1 部品に集約して登録する．検索結果のランキングには 1 部品として提示され，利用者は集約後のソースコードと元の 2 つのソースコードを閲覧可能とする．この集約方法については厳密に規定していなかったが，リファクタリングによる共通の親クラスの抽出や，共通ソースコードのテンプレートと差分パッチを想定していた．

Vol.2009-SE-166 No.20
2009/11/6

表 3 参加者別の回答数.

参加者 ID	立場	a	b	c	d	備考
A1	教員	0	3	11	4	第 2 セッションから参加.
A2	学生	0	2	20	2	
A3	教員	0	0	22	2	
A4	学生	0	1	23	0	
A5	企業	0	0	22	2	
B1	教員	5	5	7	6	
B2	学生	4	5	9	6	
B3	学生	9	2	7	6	
B4	教員	11	3	6	4	
B5	学生	10	5	7	2	
C1	教員	2	13	7	2	
C2	学生	2	14	6	2	
C3	学生	1	13	8	2	
C4	教員	1	13	6	4	
D1	教員	3	2	12	7	
D2	教員	7	1	10	6	
D3	教員	2	8	11	3	
D4	教員	5	0	12	7	
計		62	90	206	67	

(c) 関連リンクを登録する. 2 つのソースコードはそれぞれ個別の部品として登録するが，関連リンクを登録しておく．検索システムは，利用者が部品を見ているとき，関連部品情報の提示を行う.

(d) 個別の部品として登録する. 2 つのソースコードには強い関連がなく，それぞれを個別の部品として取り扱う.

回答用紙には，選択肢とともに，その選択理由を記載してもらった.

3. 実施結果

回答を参加者別に集計したものを**表 3** に，ソースコード別に集計したものを**表 4** に示す．表 4 における抽出方法の列では，使用した Twigi の閾値を示している．Twigi 0.85 という記述は，閾値 $x = 0.85$ ではこれらのクラスが同じグループに含まれるが，閾値を 0.05 増やした $x = 0.90$ では別のグループになったことを示している．つまり，そのグループのファイル間の類似度が 0.85 以上 0.90 未満であったことを意味する.

表 3 から，グループおよび参加者によって判断に偏りがある様子を知ることができる．1 つのソースコードに対して 1 グループの意見だけに注目すると，多くの場合は 1 つあるいは 2 つの選択肢に固まっており，平均では 1.55 個の選択肢が選ばれている状況であった．これは，意見が近い参加者を同一グループに含めたことが影響していると思われる.

今回収集した 426 の回答のうち，判断が議論の途中で変更されたのは 109 回で，1 人あたり最低 1 回，最高で 11 回の判断変更を行っていた．変更理由には様々なものがあるが，グループ参加者の意見を聞いてその意見に賛同した，あるいは自分の誤解に気付いて意見を修正したという理由などが見られた．グループでの議論により，参加者がより正しいと感じた回答を獲得できたと考えられる.

3.1 ソースコードの分類結果

本項では，各選択肢を選んだ参加者の意見を整理する.

(a) 一方のみを残すソースコード. (a) の判断が多数回答されたソースコードの組では，ソースコードの文面がよく似ており，対処法という点では意見が分かれるが，少なくとも何らかの関連が見出されている（選択肢 (d) が選ばれていない）．そのようなコードの代表例である P1 は，異なるパッケージに存在する同名のクラスである．一方のファイルでは，同じパッケージに存在するインタフェースを参照しているが，他方では，同名のインタフェースを内部クラスとして定義している．また，やはり (a) が多数選択されている S1 は，パッケージ名とインデントのみが異なるソースコードの組である．これに対して，機能が十分に近ければどれか 1 つだけを残していれば十分である，インデントの差異のみであれば集約する価値がない，という理由から (a) が選択されている.

(a) 以外の選択肢を選んだ参加者の意見としては，ソースコードの大部分が一致していても，ソースコードを捨てることによりパッケージ名やコメントなどの検索キーとなりうる情報や，開発者の「意図」が失われる可能性があるとの指摘があった．また，ソースコードのパッケージ名などからライブラリ名を知り，そのライブラリ全体を再利用するという再利用経路が失われるという指摘もあった．ホワイトボックス再利用のように，部品のソースコードを確認する開発者にとっては，同じ部品が重複して登録されていないほうが利便性は高まるなど，再利用形態まで考慮すべきではないかという意見が出た.

(b) 集約するソースコード. 本ワークショップでは，集約という言葉の意味を厳密に規定していなかったため，グループごとに判断は異なるが，Java のリファクタリングの概念に強く影響されいる.「親クラスとして抽出するべき」「パッケージ名だけが異なるソースコードの組は Java の言語仕様上はまとめられないので (b) にはできない」など，具体的な集約方法まで考慮した意見が見られた．また，集約の結果が検索にどのように使用されるか分からない，再利用データベースの構築という観点では部品の集約が将来の保守における利益に

表 4 ソースコードの特性と分類結果.

ソース ID	抽出方法	比較されたクラス（2 つ目のクラスのパッケージ名は，1 つ目と同一の場合省略）	a	b	c	d	備考
P1	Twigi 0.95	com.adito.applications.server.IOStreamConnector，com.maverick.multiplex.IOStreamConnector	9	4	4	0	遅刻者のデータなし.
P2	Twigi 0.90	org.apache.commons.javaflow.bytecode.transformation.asm.ResumeTestCase，InvokerTestCase	2	4	10	1	同上.
P3	Twigi 0.90	org.apache.commons.lang.builder.DefaultToStringStyleTest，StandardToStringStyleTest	2	0	11	4	同上.
P4	Twigi 0.85	org.eclipse.swt.opengl.examples.GradientTab，FogTab	0	0	4	13	同上.
P5	Twigi 0.90	org.apache.commons.lang.mutable.MutableIntTest，MutableLongTest	5	5	7	0	同上.
P6	CCFinder	portal.presentation.messenger.mmsg_msgfaxmain，mmsg_msgicqmain	2	11	4	0	同上.
Q1	Twigi 0.90	org.eclipse.ui.actions.TextActionHandler，org.eclipse.ui.internal.navigator.TextActionHandler	1	3	14	0	
Q2	Twigi 0.85	org.eclipse.jdt.internal.ui.preferences.CodeAssistPreferencePage，CodeAssistAdvancedPreferencePage	7	8	3	0	
Q3	Twigi 0.90	org.apache.commons.io.filefilter.PrefixFileFilter，SuffixFileFilter	0	8	10	0	
Q4	Twigi 0.95	phex.common.collections.IntSet，org.limewire.collection.IntSet	7	2	9	0	
Q5	Twigi 0.95	org.eclipse.core.tools.resources.metadata.MarkersDumpingStrategy_1，MarkersSnapshotDumpingStrategy_1	1	1	16	0	
Q6	Twigi 0.90	org.apache.commons.net.smtp.SMTPConnectionClosedException，org.apache.commons.net.ftp.FTPConnectionClosedException	1	4	5	8	
R1	Twigi 0.85	org.eclipse.swt.browser.DownloadFactory，ExternalFactory	5	6	6	0	「判断できない」1 名.
R2	Twigi 0.95	com.adito.security.itemactions.DisableAuthenticationSchemeAction，EnableAuthenticationSchemeAction	1	8	9	0	
R3	Twigi 0.90	org.agilewiki.elementmodel.wholeness.roots.updatecommands.DumpCommandRoot，LoadCommandRoot	0	5	9	4	
R4	Twigi 0.85	org.eclipse.jdt.internal.debug.eval.ast.instructions.OrOperator，AndOperator	0	4	14	0	
R5	Twigi 0.95	org.apache.commons.io.filefilter.OrFileFilter，AndFileFilter	1	1	16	0	
R6	Twigi 0.85	org.apache.commons.jxpath.ri.model.jdom.JDOMNamespacePointer，org.apache.commons.jxpath.ri.model.dom.NamespacePointer	2	0	16	0	
S1	Twigi 0.95	com.adito.security.pki.SimpleASNWriter，com.maverick.util.SimpleASNWriter	9	4	5	0	
S2	Twigi 0.90	org.apache.commons.io.output.PackageTestSuite，org.apache.commons.io.input.PackageTestSuite	1	2	5	10	
S3	Twigi 0.85	org.eclipse.ui.internal.ide.StringMatcher，org.eclipse.ui.views.navigator.StringMatcher	3	7	8	0	
S4	Twigi 0.90	org.eclipse.core.internal.commands.operations.GlobalUndoContext，org.eclipse.core.commands.operations.UndoContext	1	1	5	11	
S5	Twigi 0.85	org.eclipse.jdt.internal.ui.viewsupport.LibraryFilter，org.eclipse.jdt.internal.ui.filters.ContainedLibraryFilter	2	0	12	4	
S6	Twigi 0.50	org.eclipse.swt.internal.motif.XAnyEvent，XConfigureEvent	0	2	4	12	

つながらないといった理由から，(b) は選びにくいという指摘があった.

このような状況に影響されてか，共通部品を作成して登録すべきである，という意見が多数派となったソースコードは比較的少ない．P6 は，使用している文字列リテラルの `fax` と `icq` という部分だけが異なっているコードの組である．Q3 は，ファイル名に対してその `Prefix` あるいは `Sufix` のどちらを検査するかが異なっている．そのため，コメントおよび変数名の `prefix` と `sufix` を入れ替え，また，メソッド呼び出し `checkStartsWith` と `checkEndsWith` を入れ替えるだけで，一方から他方へと変換できる．R2 は，`Disabled` と `Enabled` という対になったクラスであり，`isEnabled` メソッドの戻り値の真偽が反転しているほかは，クラス名と文字列リテラルに出現する `Enable` と `Disable` が入れ替わっているだけとなっている．S3 は，コメントの英単語の綴りが一方では誤っており他方では正しいなどの特徴がある組で，同一ソースコードの異なる版である可能性が推測される．これらは，いずれも，機能がきわめて類似しているが同一ではなく，かつ，ソースコードの差分も非常に小さいという特徴があったといえる.

(c) 関連リンクを登録するソースコード. 2 つのソースコードの関連リンクだけを登録しておくという選択肢は，(a) や (b) と比べて情報を失うリスクがないことから，全回答のうち約半数の 206 個の回答が (c) という結果となった．なお，関連リンクとして，どのような種類の関連を与えるのかは規定していなかったため，参加者ごとに，たとえば派生関係やバージョンの新旧など，単なる検索用のリンク以外を想定して，違う意味で同じ判断 (c) になっている可能性がある.

(c) が多数派となった P2 は JUnit テストの組で，異なる `TestSuite` クラスの名前を JUnit に与える役割を持っている．P3 は，同じパッケージに含まれている JUnit テストのための 2 つのクラスで，テストに用いる引数の値や出力文字列が異なるほか，作成した時期

が異なるためかライセンス表記にも違いがみられる．これらのコードは，単純には，テストという同じ役割に所属していると考えることができるが，同一機能ではないために集約や破棄は難しく，関連リンクを作るという選択肢になったと思われる．その一方で，作られた時期が異なるファイルには互いに関連があるとは思えない，という意見もあった．

Q1 と Q5 は，参照する定数が異なるクラスの組である．Q1 では，異なるクラスに定義された同名のフィールドを参照しており，一方で `super(C1.Delete);` という定義があれば，他方では `super(C2.Delete);` というように，参照先だけが異なっている．

R4 と R5 は，論理演算子 AND と OR に相当する処理をプログラム中で計算するためのクラスの組である．使用する論理演算の違いや，実装上の細かい違い以外の構造はほとんど同じである．これらのコードは集約は困難であるが，対となる機能を実装したと想像することができたことから，(c) が選ばれたと考えられる．

(d) 個別の部品として登録するソースコード. 目的が異なる，あるいは，検索として個別に提示されるほど強い関係がないと判断されたソースコードは，(d) と判定されていた．

P4 は，`opengl` パッケージを使用するプログラム例の組である．画面への描画を行う構造自体は同一であるが，描画用の処理には異なる機能を使用しており，それらはまったく個別のものであると判断されている．

Q6 は，例外クラスの組である．特別な機能を持たない例外クラスであるので，所属パッケージ，クラス名とコメント以外はまったく同一である．また，S2 は apache commons ライブラリでの JUnit テストを実行するためのクラスで，参照するテストケースのクラス名だけが異なり，他は同一というソースコードの組である．Q6 や S2 は，クラス名自体がプログラム上重要であり，参加者は，それ以外のソースコード部分が似ていることにはあまり興味を示さなかったと考えられる．そもそもソースコード検索のデータベースに登録する価値がないコードであるという指摘もあった．

S6 は，`public` なフィールドばかりを持つ，C 言語でいう構造体に対応するような 2 つのクラスである．クラスとして何とか抽象化できるのではないか，という意見がある一方で，まったく同じコードには見えないなど，参加者によって意見が異なっていた．

3.2 類似性の判断基準

セッション終了後，各グループに対して，ソースコードのどのような点に注目したか，口頭でのインタビューを実施した．また，参加者が注目した要素を推定するために，回答用紙に記載された判断理由の中で頻繁に出現した単語を数え上げ，**表 5** として整理した．ただし，1 つのソースコードの判断理由に複数の単語が出現している場合もある．

表 5 回答用紙に出現した単語.

合計出現回数	使用人数	単語カテゴリ	単語
40	10	バージョン管理	バージョン (16)，版 (7)，派生 (8)，新 (8)，オリジナル (1)
31	12	クラス階層	継承 (5)，extends(5)，子 (13)，親 (8)
28	9	クラスの関係	対 (28)
25	14	テスト	テスト (25)
22	7	ロジック	ロジック (14)，アルゴリズム (8)
16	4	コード例	例 (6)，サンプル (5)，example(5)
14	9	コメント	コメント (3)，Javadoc(11)
13	5	識別子	識別子 (9)，名 (4)
10	6	役割	役割 (10)
7	5	振舞い	振舞い (7)

これらの内容を分析した結果，参加者がソースコードの類似性を判断した基準には，以下の項目が含まれていると考えられる．

識別子の類似性. グループ A，B は，使用されている識別子が似ているソースコードには関連がありそうだと考えていた．

クラスの振舞いの類似性. 振舞いがまったく同じであれば，集約する必要はなく一方だけを使用すればよい，あるいは，実装例として 1 つあればよいという意見があった．特に動作を分析していたのはグループ C で，コメントはあまり参考にしなかったとインタビューで回答している．また，2 つのソースコードから共通の親クラスを抽出するなど，一見して共通化することが可能と見えるほどに動作が似ているコードについては，参加者は集約方法を意識していた．

クラスの作成目的の一致. 本ワークショップで，全参加者が (b) あるいは (c)，つまり関連性があり，かつ両方を関連させたまま残すべきと判定したソースコードは，Q3 と R4 の 2 つであった．これらは，`Prefix` と `Suffix`，`And` と `Or` というように，一対となる概念が 1 つの機能のために実装されたと推測することが容易なコードである．

クラスの役割. たとえソースコードが非常によく似ていても，参加者らは似ていること自体に興味を示さなかったコードとして，例外処理のためのクラスや JUnit テストのクラスが挙げられる．一部の参加者は，これらのコードを「自動生成されたようなコード」と表現していた．

ファイルの新旧. グループ B，D は，コメントに記載された日付情報などからソースコードの派生関係を考慮しようとしていた．一方のバグが他方では修正されたと思えるようなバージョンの差異では，古いほうを捨てるなど，集約とは異なる対応が必要であると

いう指摘があった．このことから，単に似ているかどうかだけでなく，派生関係が重要な意味を持つことがうかがえる．

参加者は，ソースコードの類似性を判断する基準として，識別子などの見た目だけでなく，振舞いや役割の類似性やクラス階層での位置づけ，ファイルの派生関係といった幅広い情報を用いていたことが判明した．この結果を踏まえると，類似したソースコードを調査するためのツールを開発する場合は，単に該当するソースコードそのものを提示するだけでなく，ソースコードが置かれた状況を効果的に利用者に提示する機能が重要であると考えられる．また，本ワークショップでは，ソースコード再利用というコンテキストに限ったにも関わらず，ソースコードの類似性の判断が参加者によって大きく異なるものとなった．ソースコードの類似性判定に関する技術の適用実験を行う際には，その妥当性を注意深く検討する必要があるといえる．

4. 妥当性への脅威

参加者募集にあたっては，過去に SIGSE ウィンターワークショップ「プログラム解析」グループへと参加したことのある研究者を中心に呼びかけを行ったため，ソフトウェア再利用，部品検索，リファクタリングなどの作業コンテキストに関しては十分な知識を持った参加者が数多く含まれている．ただし，参加者の多くが教員および学生であることから，実務者の意見とは差がある可能性がある．

参加者は，アンケートにおいて練習問題への回答理由を述べているが，判断基準を完全には明文化していない．そのため，参加者が自分自身の判断基準を途中で変更している可能性がある．また，共同作業における議論には時間がかかり，参加者の疲労が大きかったと推測される．第 3 セッション，第 4 セッションでは，疲労した参加者による短絡的な判断が含まれる可能性がある．

ソースコードの提示方法として，ソースコードの組を対象とし，それ以外の情報，たとえばファイルの日付情報や，どのプロジェクトから取得したファイルであるかという情報は，参加者に提供しなかった．しかし，コメント中の日付に注目した参加者や，類似クラスがただ 2 つだけなのかそれより多いかによっても判断が変わりうると指摘した参加者もあり，ソースコードの提示方法は結果に影響していると考えられる．

提示したソースコードは，ツールがある一定の類似性を持つと判定したソースコードのみであった．そのため，ソースコードの文面は似ていないが開発者が類似していると考えるようなソースコードが存在する可能性については調査を行っていない．

本ワークショップでは，作業コンテキストとして再利用データベースの構築を選んだ．検索システムにおけるソースコード類似性のチェックという状況に限れば，同じように使える部品が大事であるというように，作業コンテキストは結果に強く影響を与えていると考えられる．しかし，再利用といっても，部品をそのままブラックボックスとして再利用するのか，改変して再利用するのかで変わってくる，また，クラスの再利用かコード片の再利用でも基準が変わった可能性がある，といった指摘があった．参加者の中には，再利用データベースにおける検索方法なども判断材料にしたいという意見もあり，作業コンテキストの設定は，厳密に達成されていたわけではない．

5. 実施における反省点

本ワークショップは，著者らにとって初めての共同作業型のワークショップであった．本節では，実施途中に気付いた点や，今後，類似した共同作業型ワークショップを開催する際に改善可能であると思われる点を整理する．

5.1 作業コンテキストの設定

作業コンテキストの提示方法に関しては，本ワークショップでは，ソースコードの「集約」という用語が曖昧であり，Java 言語に固有の範囲であるのか，あるいはその他の方法を含むのか，参加者によって判断が分かれる結果となってしまった．

また，ソースコードを検索できるようにするという作業目的から，検索アルゴリズムに対する前提について，たとえば集約したときに差分データは検索対象となるか，検索時の索引に影響するのか，などの細かい条件に関する質問を行う参加者もあった．どこまで詳細化しておくかには議論があるが，本ワークショップでは，少なくとも「集約」という用語に対する一貫した定義を用意しておらず，事前の検討が十分でなかったといえる．

作業状況の詳細を細かく詰めるほど，参加者にとっては判断の揺れが少なくなり，実験の再現性を向上できる可能性がある．一方で，事前に詳細な作業状況を作ることは難しく，予備実験の実施が重要となると思われる．

5.2 対象ソースコード

本ワークショップでは，30 分〜45 分程度を 1 セッションと区切って 1 セッションあたり 6 組のソースコード，4 セッションを実施した．参加者が閲覧するべきソースコードの分量が多く，第 2 セッションあたりでは議論が盛り上がっていたが，第 3 セッション，第 4 セッションでは低調となっており，疲労の影響が出ていたと思われる．

本ワークショップでは，ソースコードの提供方法として，行単位の差分と，行内の差分情

報を WinMerge によって可視化したものを印刷して提供した．しかし，行単位での diff を基本として提示したため，インデントのみが異なるようなソースコード組に対して，それらが互いに大きく変更されているかのように可視化されてしまった例が 1 組存在した．参加者の負担を下げるために，ソースコードの差異がたとえばパッケージ名，クラス名，メソッドの本文である，というように，注釈を追加するというやり方も考えられる．ただし，そのような注釈によってソースコードを実際に目で見比べる時間が短くなり，実験結果に影響を与える可能性もある．

5.3 セッションの進行方法

セッションの時間進行については各グループに委ねたところ，予定時間を大きく超過する傾向があった．進行速度が最も安定していたグループは，アラームを用意して全員が同時に 1 つのソースコードを読み，判断を話し合うという方式を採用していた．

本ワークショップでは，残念ながら，議論の経過の様子を十分に記録することができなかった．回答用紙のみからでは，各個人の判断は記載されているが，断片的な単語のみが記載されていることもあり，たとえば，誰の意見に影響されたのか，合意が生じたのか，あるいは偶然判断が一致しただけなのか，といった詳しい状況を知ることができなかった．各グループでの議論を記録する（発言は行わない）書記が 1 名ずつ加わることが効果的であると考えられる．また，本ワークショップでは，事後アンケートを口頭で実施したが，全員からの意見を十分に収集できなかった可能性がある．各個人で，ソースコードのどの部分に注目したかなど，より詳細な情報をアンケートに記載してもらうことが情報収集として効果的であった可能性がある．

今回，ソースコードと回答用紙を紙で用意したところ，ソースコード 4 部を印刷したものが A4 用紙で 456 枚，回答用紙が A4 で約 400 枚必要となった．著者らはすべての資料を会場まで持参したが，ソースコードの量や印刷方法などによっては，ローカルアレンジ担当者との連携による郵送などの手段が必要になる可能性がある．

最後に，当日参加や，やむを得ない事由により途中参加となってしまった参加者への対応が必要であった．簡単な対策としては，当日参加者を 1 つのグループにまとめる方法があるが，配布資料の印刷部数などに影響することから，既存グループに吸収する形とした．当日参加者には，口頭で開発経験など自己紹介を行ってもらったが，十分な情報を漏れなく獲得するために，参加者と同様のアンケートを記入してもらうべきであった．なお，SIGSE ウィンターワークショップのように当日参加を禁止した形式での実施であれば，参加者のグループ分けや配布資料の印刷等，事前準備の負担が大きく削減されると考えられる．

6. ま と め

ソースコードの類似性ワークショップでは，18 名の参加者を 4 つのグループに分け，ある 1 組のソースコードが類似していると考えるかどうか，という判断を手作業で行っていく共同作業を実施した．その結果，使用されている識別子の類似性やクラスが作成された目的の類似性，派生関係などが判断に関係していることを確認した．

ワークショップで使用したデータ，分析結果等はすべてインターネット上で公開している[10] ので，自由に活用していただければ幸いである．

謝辞 ワークショップにて共同作業にご参加いただきました 18 名の参加者に，深く感謝いたします．

参 考 文 献

1) Bellon, S., Koschke, R., Antoniol, G., Krinke, J. and Merlo, E.: Comparison and Evaluation of Clone Detection Tools, *IEEE Transactions on Softare Engineering*, Vol.33, No.9, pp.577–591 (2007).
2) CCFinder Official Site: http://www.ccfinder.net/.
3) Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code, *IEEE Transactions on Software Engineering*, Vol.28, No.7, pp.654–670 (2002).
4) Roy, C.K. and Cordy, J.R.: Scenario-Based Comparison of Clone Detection Techniques, *the 16th International Conference on Program Comprehension*, Los Alamitos, CA, USA, IEEE Computer Society, pp.153–162 (2008).
5) WinMerge: http://winmerge.org/.
6) 澤 健一，肥後芳樹，楠本真二：コードクローン検出ツールを用いた不具合検出手法の提案と評価，電子情報通信学会技術研究報告， Vol.108, No.173, pp.67–72 (2008).
7) 吉田則裕，肥後芳樹，神谷年洋，楠本真二，井上克郎：コードクローン間の依存関係に基づくリファクタリング支援，情報処理学会論文誌， Vol.48, No.3, pp.1431–1442 (2007).
8) 渡邉卓也，増原英彦：類似プログラムの提示ツール Selene，日本ソフトウェア科学会第 24 回大会 講演論文集，3B-2，pp.1–9 (2007).
9) 佐々木裕介：利用関係に基づく類似度を用いた Java コンポーネント分類ツールの作成，大阪大学 卒業論文, http://sel.ist.osaka-u.ac.jp/~lab-db/Bthesis/contents.ja/121.html (2009).
10) 石尾 隆，山本哲男：ソフトウェアの類似性ワークショップ，http://sel.ist.osaka-u.ac.jp/JWSCS2009/.