ホワイトボックス単体テストにおけるペアテスティング

Pair Testing for White Box Unit Testing

坂本 一憲 * **本田 澄** † **鷲崎 弘宜** ‡ **深澤 良彰** §

あらまし ペアプログラミングはソフトウェアの品質を向上もしくはペアの相方に対する教育や知見の共有効果があることが分かっている．ペアプログラミングは 2 人組のエンジニアがソフトウェアを実装する手法であり，テストの工程に適用する場合はペアテスティングと呼ばれている．しかし，既存研究ではテストの工程に限定したペアテスティングに関する議論がほとんどされておらず，実際にペアテスティングによってどの程度欠陥の検出能力が向上するのか分かっていない．本論文では，1) 2 人組のペアが 2 台のパソコンで協力をしない，2) 2 人組のペアが 1 台のパソコンで協力をする，3) 2 人組のペアが 2 台のパソコンで協力をする，3 種類のテストコードの記述方法について，被験者による比較実験を通してどの方法が最もテストの品質に良い効果を与えるかを明らかにする．36 名 18 組に対する実験の結果，概してテストコードの品質は，3) 2 台で協力する場合に最も良く，2) 1 台で協力する場合に最も悪かった．

Summary. Pair programming is known as an effective way to improve software quality and share knowledge with each other, teach technologies with each other. Pair programming is a way that a pair of two programmers does programming, pair testing, on the other hand, is a way that a pair of two programmers does testing. However, existing researches did not discuss about pair testing which is applied to only testing, thus, the effectiveness about bug detection ability of pear testing has yet to be revealed. In this paper, we compare three testing ways: 1) a way that a pair with two PCs without collaborating writes test code 2) a way that a pair with a PC with collaborating writes test code 3) a way that a pair with two PCs with collaborating writes test code. As a result of our experiment with 36 students, overall, the third way results highest test quality and the second way results lowest test quality.

1 はじめに

ペアプログラミングはエクストリーム・プログラミング（XP）の 1 手法であり [1]，開発者 2 人組のペアが 1 台のパソコンを共有してプログラミングを行うことで，2 人の開発者が個々人でプログラミングを行うよりも，短期間で高いソフトウェア品質を得ることができる [2]．ペアプログラミングは主に製品コード [1] の実装を対象としているが，リファクタリングやソフトウェアテスト（以降，テスト）を含めたソフトウェア開発全体の活動を対象とすることがある [2,3]．なお，ペアプログラミングと同様の方法で行うソフトウェアテストはペアテスティングと呼ばれている [2]．

これまで，ペアプログラミングに関して様々な研究がなされてきたが [2–12]，我々が調査した限りペアテスティングのみに特化した研究は行われておらず，ペアテスティングの有効性や有効性の高い方法に関する知見が明らかになっていない．そこ

*Kazunori Sakamoto, 国立情報学研究所

†Kiyoshi Honda, 早稲田大学

‡Hironori Washizaki, 早稲田大学

§Yoshiaki Fukazawa, 早稲田大学

[1]本論文ではテスト対象のプログラムのソースコードを製品コードと呼び，製品コードをテストするコードをテストコードと呼ぶ．

で，本論文の目的は，ペアテスティングがテストの品質にどのような影響を与えるか，また，どのようなペアテスティングの方法が最も良いテスト品質を実現するか明らかにすることである．

テストの品質を評価するための手法として，テストカバレッジが広く利用されている [13]．テストカバレッジは，テストによってテスト対象の製品コードがどの程度網羅的に実行されたかを表す指標であり，テストカバレッジの測定値が高ければ高いほど，欠陥の検出能力が向上することが分かっている [14, 15]．テストカバレッジには様々な種類が存在しており，例えば，製品コード中のステートメントのうちテストで実行されたステートメントの割合を示すステートメントカバレッジや，製品コード中の if 文などの分岐を表す要素（ステートメントおよび式）のうち全ての分岐先を実行された要素の割合を示すブランチカバレッジなどがある．

ミューテーション解析は，必要な計算時間が膨大なため学術界での利用が主であるが，テストの品質を評価するための手法として利用されている [16]．ミューテーション解析は，製品コード中にバグを埋め込んだミュータントと呼ばれる亜種プログラムを機械的に複数生成して，評価対象のテストがミュータントが含むバグを検出できるかどうかを検証する手法である．実際に人が埋め込んだバグとミューテション解析によって埋め込んだバグを比較して，ミューテーション解析がテストの欠陥検出能力を測る上で有効であることが分かっている [17]．

本論文では，ペアテスティングがホワイトボックス単体テストにおいて有効であることを確認するため，**方法 1（ソロ 2 台）**：2 人組のペアが 2 台のパソコンで協力せずにテストコードを記述する方法，**方法 2（ペア 1 台）**：2 人組のペアが 1 台のパソコンで協力してテストコードを記述する方法，**方法 3（ペア 2 台）**：2 人組のペアが 2 台のパソコンで協力してテストコードを記述する方法について，被験者による比較実験を通して，どの方法で最も高い品質のテストコードを記述できるかを明らかにする．なお，Sajeev らの研究では，1 台のパソコンの共有がペアに好まれないことが報告されているため [12]，2 台のパソコンでペアプログラミングのように議論しながらテストコードを記述するペアテスティングの方法も比較対象に加えた．

早稲田大学基幹理工学部情報理工学科の大学 4 年生，および，同大学院基幹理工学研究科情報理工学専攻の大学院生 36 名 18 組に対して，3 種類のプログラムを用意して，それぞれ 18 分間で上述の 3 種類の方法でホワイトボックス単体テストにおけるテストコードを記述させる実験を実施した．用意したプログラムと実験で記述したテストコードに対して，テストカバレッジ測定とミューテーション解析を実施して，得られたテストカバレッジの測定値とミューテーション解析においてテストが検出したバグの数に基いて，上述の 3 種類の方法を比較した．

その結果，概して，**方法 3（ペア 2 台）**を利用する場合にテストコードの品質が最も良く，**方法 2（ペア 1 台）**を利用する場合にテストコードの品質が最も悪かった．ただし，**方法 1（ソロ 2 台）**と**方法 3（ペア 2 台）**の差はほとんど見られず，取り組む課題の内容が難しく取り組む期間が短い場合は**方法 1（ソロ 2 台）**の方が良かった．したがって，課題の難易度に対して，十分に課題に取り組む期間が長い場合は，**方法 3（ペア 2 台）**が最も良い結果であるということが分かった．また，記述されたテストコードの品質は最も悪かったものの，**方法 2（ペア 1 台）**が最も多くのペアから好まれた方法であった．

本論文の研究課題（Research Questions; RQs）は以下のとおりである．なお，本論文では，ホワイトボックス単体テストにおけるテストコード記述を対象とする．

- RQ1: 方法 1（ソロ 2 台）と方法 2（ペア 1 台），方法 3（ペア 2 台）のいずれの方法が，最も高い品質のテストコード記述を実現できるか？
- RQ2: 方法 1（ソロ 2 台）と方法 2（ペア 1 台），方法 3（ペア 2 台）の 3 種類の方法に，対象とするプログラムによって向き不向きは存在するか？
- RQ3: 方法 1（ソロ 2 台）と方法 2（ペア 1 台），方法 3（ペア 2 台）のいずれの方法が，最も多くのペアに好まれるか？

本論文の貢献点は以下のとおりである．

- ペアプログラミングに対する既存研究が十分に行われているのに対して，ペアテスティングに対する研究や調査がほとんど行われていないことを明らかにした点．
- 1 台のパソコンを共有することに対する批判を踏まえて，協調的にテストコードを記述するか否かと，1 台のパソコンを共有するか否かを考慮して，3 種類の方法で比較実験を行った点．
- 今回実施したペアテスティングの比較実験においては，2 台のパソコンを活用したほうが 1 台のパソコンを共有するよりも，高い品質のテストコードの記述を実現することを明らかにした点．
- 本実験では，対象とするプログラムや課題に取り組む期間によっては，ペアが協調せずにテストコードを記述したほうが良いケースがあることを明らかにした点．
- 本実験では，記述されたテストコードの品質は最も悪かったものの，**方法 2（ペア 1 台）**が最も多くのペアから好まれた方法であったことを明らかにした点．

本論文の構成は次のとおりである．1 節において，はじめにとして本論文の概要を説明し，2 節において，関連研究について，3 節において，比較対象となるペアテスティングに関連する 3 種類のテスト方法について説明する．本論文で行う実験については，4 節において，被験者による比較実験の手順と，対象とするプログラムについて，また，被験者のテストに関する知識について説明を行う．5 節において，結果として，テストカバレッジとミューテーション解析について，また，アンケートについて示す．最後に，6 節において，本論文における妥当性への脅威を示し，7 節において，まとめと今後の展望を示す．

2　関連研究

現在，ペアプログラミングはエクストリーム・プログラミング（XP）の 12 種類のプラクティスの 1 プラクティスとして広く普及しているが [1]，XP が提唱される以前から類似する手法の提案はあった．例えば，Nosek らは，1995 年にペアでプログラミングを行ったほうが生産性が高く，かつ，バグの少ない高品質なソフトウェアが作成できるという報告をしている [18]．

ペアプログラミングの有効性については様々な研究がなされており，製品が出荷されるまでに必要な期間，開発コスト，ソフトウェアの品質に対する影響のみならず，プログラミング初学者に対する教育効果や，モチベーションの維持，信頼関係の改善，知識の共有などへの影響について議論がされている [2–12]．

Arisholm らは，一概に生産性が向上するというわけではなく，対象となるプログラムの複雑性とプログラマーの習熟度によって効果が変化すると述べており，彼らの 295 名に対する大規模な実験では，初学者が複雑なプログラムに対するプログラミングにおいて誤りが減ることと，中級者以上が簡単なプログラムに対して正確にプログラミングを行う際に必要な時間が減るということにおいて，効果が得られたと報告している [8]．

Hannay らは，既存研究を調査することで，ペアプログラミングの効果について議論をしており，既存研究の結果からは研究間の相違が大きく，結果にバイアスがかかっている可能性が高いことを指摘している [9]．しかし，対象となるプログラムの複雑性とプログラマーの習熟度について考慮した場合，複雑なプログラムに対しては品質向上の効果があり，簡単なプログラムでは必要な時間が上昇する効果があると主張している．また，ペアプログラミングによって開発コスト削減と品質向上の両方を達成することは難しく，トレードオフの関係であると主張している．

Salleh らは，ペアプログラミングに従事するプログラマ個人の特徴（パーソナリティ）と有効性の相関性を調査して，ペアプログラミングという経験に対する開放

性，誠実性，外向性，同調性，神経症的傾向について，開放性のみがペアプログラミングの有効性と顕著な相関があったことを報告している [10].

Sillitti らは，ペアプログラミングの有効性を測定する際のバイアスをできるかぎり抑えるため，開発者の活動，具体的には開発作業中のソフトウェア利用履歴を解析することで，ペアプログラミングを導入した方が開発に関連する作業への集中力が高まることを報告している [11].

Sajeev らは，144 名の学生に対して実験を実施して，Q1) ペアプログラミングにおいて好ましくない特徴，Q2) テストケースを用いた品質評価と参加者のアンケート調査を通したペアプログラミングの有効性，Q3) ペアプログラミングを練習する際に簡単な問題と複雑な問題のどちらを先に取り組むべきかについて調査を行った [12]．その結果，A1) 1 台のパソコンで画面やキーボードなどの入出力機器を共有して，ドライバーとナビゲーターの役割を入れ替えながら進めるという特徴を最も好ましくないと考えており，A2) テストケースを用いた品質の評価ではペアプログラミングは複雑な問題を解くときのほうが簡単な問題を解くよりも効果的であったが，一方で，アンケート調査では簡単な問題を解く方が良いという結果が得ら，また，A3) ペアで簡単な問題を解いてから複雑な問題を解く方が効果的であるという結果が得られたと報告している．

しかし，我々が調査した限りでは，ペアプログラミングに対して様々な研究が存在しているものの，テストコードを記述する際にペアを組むペアテスティングについて調査している研究はなかった．したがって，本論文では，ペアテスティングの有効性に関する調査研究の第一歩として，単純なペアテスティングの方法について，比較的規模の小さいプログラムに対して被験者による比較実験を実施する．

3　比較対象となるペアテスティングに関連する 3 種類のテスト方法

現時点では，ペアテスティングに関する十分な文献は存在しておらず，ペアテスティングの具体的な方法論に関してはインターネット上でいくつかの情報が発見されるだけである [2] [3] [4]．これらの情報の中には，テストコードを記述できるプログラマとソフトフェアの仕様を熟知したビジネスアナリストが協調すると良いとするものがあり，ペアプログラミングが知見の共有においても有効であることから [19]，一定の効果があると考えられる．

しかし，そもそも，ペアテスティングに関して方法や有効性が十分に議論されていないため，本論文では最もシンプルだと思われるペアテスティングの方法に対して被験者による比較実験を行う．具体的には，2 人組の開発者が 1 台のパソコンを共有して開発する方法をペアテスティングとして，2 人組の開発者が 2 台のパソコンで作業分担に関する議論のみを行いながら別々にテストをする方法と比較を行う．また，Sajeev らによる，1 台のパソコンで入出力機器を共有することを好まないペアが多いという報告を考慮して [12]，2 人組の開発者が 2 台のパソコンでテストに関する議論をしながら別々にテストをする方法も合わせて比較を行う．

テストカバレッジやミューテーション解析の既存研究によれば，製品コードを網羅的に実行するようなテストであり，かつ，実行によって出力される値を網羅的にチェックするようなテストが，バグの検出能力が高く品質の高いテストコードであると言える [20,21]．ペアで議論しながらテストコードを記述することで，コーナーケースに対するテストの漏れを防いだり，テストされていない製品コードの箇所が現れなかったりする効果が期待される．

---

[2] http://en.wikipedia.org/wiki/Pair_testing (7 月 23 日参照)

[3] http://www.slideshare.net/RehanShahKhan/rehan-pair-testing-final (7 月 23 日参照)

[4] http://softwaredevelopmentisfun.blogspot.jp/2008/07/pair-testing.html (7 月 23 日参照)

4　被験者による 3 種類のテスト方法の比較実験

本節では，ペアテスティングに関する上述の 3 種類のテスト方法を比較する実験において，まず，実験手順の詳細について述べた後，次に，被験者のテストに関する知識がどのようなものであったかということを述べ，最後に，テストコードを記述する対象のプログラムについて説明する．

4.1　被験者による比較実験の手順

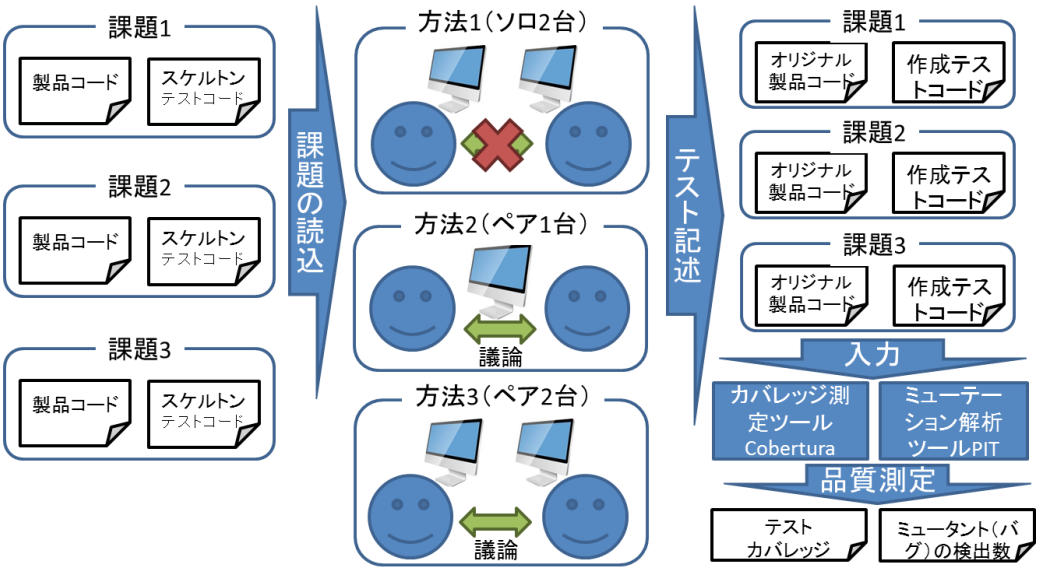

図 1　被験者による比較実験の概要

図 1 で被験者による比較実験の概要にを示す．実験ではペアを 3 つのグループ A,B,C に分けて，それぞれ 3 種類の課題プログラムに対して，3 種類の方法でテストコードを記述させた．課題プログラムは製品コードとサンプルとなるスケルトンテストコードから構成されている．課題の詳細は次節で述べる．被験者が記述したテストコードを受け取り，テストカバレッジの測定とミューテーション解析を行い，テストコードの品質を測定する．

どのグループも課題 1 から順に 3 種類の課題に取り組むが，ある課題に対して 3 つのグループがそれぞれ異なる種類の方法で取り組むこと，各グループが 3 種類の課題を通して各方法に 1 回ずつ取り組むことを満たすように，被験者の実験手順を決めた．表 1 で各グループが各課題に対してどの方法で取り組むかを示す．

表 1　実験でのグループ表

グループ表	**課題 1**	**課題 2**	**課題 3**
グループ A	**方法 1（ソロ 2 台）**	**方法 3（ペア 2 台）**	**方法 2（ペア 1 台）**
グループ B	**方法 2（ペア 1 台）**	**方法 1（ソロ 2 台）**	**方法 3（ペア 2 台）**
グループ C	**方法 3（ペア 2 台）**	**方法 2（ペア 1 台）**	**方法 1（ソロ 2 台）**

図 1 で示す実験手順について，以下で詳細を述べる．

1. 事前の講義で座学と演習を通して Eclipse と JUnit の使い方，および，テスト

技法に関して説明する．
2. テストコードを記述する対象プログラムを Eclipse のプロジェクトとして提供して，被験者のパソコンにインポートする．
3. 2 人で一組のペアを組んでもらい，ペアでテストコードを記述する旨を伝える．また，被験者には良いテストコードを書くことで競うように，テストコンテストという形で実験内容を伝える．
4. ペアを 3 つのグループに分けた上で，グループごとに印刷した資料を配布して，3 種類の課題プログラムについてどの方法を実施するかを説明する．
5. 各課題プログラムについて 18 分間でグループごとに指定された方法でテストコードを記述する．なお，課題プログラムとしてプログラムの仕様について説明するコメントを付けた製品コードと，JUnit の使い方で悩まないようなサンプルのテストコードを被験者に提供する．
6. 作成したテストコードとどの方法が一番良かったかという感想をメールで受け取り，テストカバレッジの測定とミューテーション解析による検出能力の測定を Cobertura と PIT を用いて実施する．

4.2 テストコードを記述する対象のプログラム

テストコードを記述する対象として，以下の 3 種類の課題プログラムを用意した．

- **課題 1:** 与えられた整数列に対して，与えられた要素が存在するかどうか判定を行うクラスを用意して，クイックソートでソートを行うコンストラクタと，二分探索により要素を探索するメソッドを対象とした．なお，クイックソートおよび二分探索の実装は，Java によるアルゴリズム事典にて掲載されたソースコードを利用した [5]．
- **課題 2:** Java 言語の標準ライブラリとして提供されている String クラスの replace メソッドと split メソッドについて，正規表現に関する機能を全て削除した上で，独自に用意した文字列ユーティリティクラスの static メソッドを対象とした．なお，String クラスの replace メソッドと split メソッドの実装は，オープンソースな Java 言語の処理系である OpenJDK を利用した [6]．
- **課題 3:** 二分探索木を用いてコレクションクラスを用意して，文字列の登録・削除・検索を行うメソッドを対象とした．課題 1 と 2 がテスト対象の機能が 2 つであるのに対して，課題 3 のみ 3 種類の操作が対象となっている．なお，二分探索木の実装は，Java によるアルゴリズム事典にて掲載されたソースコードを利用した [7]．

3 種類の課題プログラムの製品コードのステートメント数を JavaNCSS で測定したところ [8]，それぞれ 57 ステートメント, 44 ステートメント, 65 ステートメントであり，**課題 3** のプログラム規模が最も大きかった．

3 種類のプログラムの仕様に関しては，事前に一切情報を与えず，製品コードのコメントで使用に関する情報を実験に与えた．なお，実験中は任意の質問を受け付けて，仕様に関する質問に関してのみ被験者全員に対してその内容と回答を伝えた．

4.3 被験者のテストに関する知識

実験を行う事前準備として，ソフトウェアテストに関する技法，および，テスティングフレームワーク JUnit を用いた単体テストの実施について被験者に対して講義を行った．具体的には，ブラックボックステスティングについて，同値分析，境界値分析，デシジョンテーブルの考え方を説明して，それぞれの手法について演習課

[5] http://oku.edu.mie-u.ac.jp/ okumura/java-algo/

[6] http://openjdk.java.net/

[7] http://oku.edu.mie-u.ac.jp/ okumura/java-algo/

[8] http://javancss.codehaus.org/

題を通して，網羅的に様々な値を使ってテストする重要性について説明した．その後，ホワイトボックステスティングについて，FizzBuzz 問題を例にあげて，JUnit を用いたテストコードの記述方法を説明して，実際に演習として，Eclipse と JUnit を用いて FizzBuzz 問題におけるテストコードの記述を被験者に行わせた．なお，テストカバレッジについて説明を行い，製品コードを網羅的に実行するテストが良いテストであること，ミューテーション解析についても説明を行い，欠陥を検出できるテストが良いテストであるということを説明した．一方，ペアプログラミングやペアテスティングに関する説明は一切行わず，単に協力してテストコードを記述するということを実験手順として伝えた．

5 結果

本節では，実験結果について，テストカバレッジとミューテーション解析とアンケートの 3 つの観点から 3 種類の方法の比較を行う．

5.1 テストカバレッジ

表 2 でステートメントカバレッジ，表 3 でブランチカバレッジの測定結果を示す．

表 2 ステートメントカバレッジの測定結果

ステートメントカバレッジ	**課題 1**	**課題 2**	**課題 3**
方法 1（ソロ 2 台）	89.143(A)%	92.167%(B)	67.600%(C)
方法 2（ペア 1 台）	71.667%(B)	92.400%(C)	68.000%(A)
方法 3（ペア 2 台）	89.000%(C)	92.429%(A)	57.500%(B)

表 2 から，**課題 1** に関しては**方法 2（ペア 1 台）**が最も低いステートメントカバレッジを示しており，**方法 1（ソロ 2 台）**が最も高いステートメントカバレッジを示していることが分かる．また，**課題 2** に関しては，**方法 2（ソロ 2 台）**が最も低いステートメントカバレッジを示しており，**方法 3（ペア 2 台）**が最も高いステートメントカバレッジを示していることが分かる．しかし，**課題 2** に関しては，すべての方法においてステートメントカバレッジについて大きな違いは見られず，最大と最小でおおよそ 0.26% の違いしかない．**課題 3** では，**方法 3（ペア 2 台）**が最も低いステートメントカバレッジを示しており，**方法 2（ペア 1 台）**が最も低いステートメントカバレッジを示している．

この測定結果の違いは，**課題 3** の複雑さおよび規模の大きさに起因していると考えられる．**課題 3** は他の課題に比べてテスト対象が多く難しい課題であり，ペアで製品コードを理解する際に，他の課題よりも時間がかかり十分にテストコードを記述することができなかったことが考えられる．これにより，相互に確認をとる必要のない**方法 1（ソロ 2 台）**が時間的に余裕があり，他の方法よりも製品コードを理解した上でテストコードを記述することができ，その結果，最も高いステートメントカバレッジが得られたと考えられる．

表 3 ブランチカバレッジの測定結果

ブランチカバレッジ	**課題 1**	**課題 2**	**課題 3**
方法 1（ソロ 2 台）	85.714%(A)	81.000%(B)	57.200%(C)
方法 2（ペア 1 台）	70.000%(B)	78.400%(C)	53.143%(A)
方法 3（ペア 2 台）	86.000%(C)	80.714%(A)	40.833%(B)

表 3 から，**課題 1** と **課題 2** に関しては，**方法 2（ペア 1 台）**が最も低いブラン

チカバレッジを示しており，**課題 1** では**方法 3（ペア 2 台）**が，**課題 2** では，**方法 1（ソロ 2 台）**が最も高いブランチカバレッジを示していることが分かる．しかし，**課題 3** については**方法 3（ペア 2 台）**が最も低いブランチカバレッジを示しており，**方法 1（ソロ 2 台）**が最も高いブランチカバレッジを示している．この測定結果の違いは，ステートメントカバレッジと同様に，**課題 3** の複雑さおよび規模の大きさに起因していると考えられる．

5.2 ミューテーション解析

表 4 で，ミューテーション解析におけるミュータントの検出率を示す．

表 4 ミューテーション解析におけるミュータントの検出率

ミュータントの検出率	**課題 1**	**課題 2**	**課題 3**
方法 1（ソロ 2 台）	81.286%(A)	78.333%(B)	61.000%(C)
方法 2（ペア 1 台）	64.000%(B)	74.600%(C)	57.143%(A)
方法 3（ペア 2 台）	83.200%(C)	79.000%(A)	50.000%(B)

表 4 から，**課題 1** と**課題 2** に関しては，**方法 2（ペア 1 台）**が最も低いミュータントの検出率を示しており，**方法 3（ペア 2 台）**が最も高いミュータントの検出率を示していることが分かる．しかし，**課題 3** については**方法 3（ペア 2 台）**が最も低いミュータントの検出率を示しており，**方法 1（ソロ 2 台）**が最も高いミュータントの検出率を示している．ミュータントの検出率に関しても，ステートメントカバレッジやブランチカバレッジ同じような傾向の結果を示しており，テストカバレッジとミュータントの検出率に関する結果の相違点はわずかなものであると考えられる．

なお，**課題 3** についてのみ**方法 3（ペア 2 台）**が最も低い点に関しては，前節と同様に**課題 3** の複雑さおよび規模の大きさに起因していると考えられる．しかし，複雑さおよび規模の大きさに起因するという確かな結果があるわけではないため，今後，ペアがテストコードを書く様子を監視したり，もしくは，Sillitti らのように [11]，アクティブなツールのウィンドウを調査することで，**課題 3** についてのみ**方法 3（ペア 2 台）**の結果が悪い理由を調査したい．

5.3 アンケート

表 5 で各ペアに対するどの方法が最も良かったかという質問へのアンケート結果についてに示す．

表 5 どの方法が最も良かったかという質問へのアンケート結果

	ペア数
方法 1（ソロ 2 台）	0
方法 2（ペア 1 台）	10
方法 3（ペア 2 台）	2
特になし (無回答)	6
合計	18

上記のテストカバレッジとミューテーション解析による比較結果とアンケートによる比較結果を比べると，あまり効果的でなかった**方法 2（ペア 1 台）**が最も好まれ，効果があった**方法 1（ソロ 2 台）**が最も好まれていないという，興味深い結果となった．この結果に関しては，今後，被験者にインタビューを行うことで，テスト品質との相関性について調査をしていきたい．

6　妥当性への脅威

内的妥当性への脅威として，参加者のテストに対する習熟度と被験者のパーソナリティの偏りが考えられる．事前の座学や演習などの講義を通して，明らかにソフトウェアテストに習熟した被験者も数名存在しているものの，ほとんどの被験者はそもそもテストに関する知識を有しておらず，特に JUnit を利用したテストコードの記述は初めて体験する者のレベルであった．一方，被験者のパーソナリティの偏りに関しては，講義を通して明らかになっておらず，今後の研究において，被験者のパーソナリティも考慮した比較実験を行う予定である．

外的妥当性への脅威として，被験者は大学生および大学院生であり，ほとんどがテストに関する知識をもともと有していなかったため，産業界において同様の結果が得られるかわからない脅威が考えられる．ただし，学生であってもソフトウェアテストに知識を有しないプログラマ初心者と大差ないと考えられ，また，多くの既存研究では教育機関における被験者実験が多数なされていることから [2,4–7,10,12]，本論文による結果をペアテスティングの最初の第一歩と位置づけ，今後の研究で明らかにしていきたい．

7　まとめと今後の展望

本論文では，ペアテスティングの有効性および有効性の高い方法を明らかにするために，ホワイトボックス単体テストにおいて二人組のペアで 3 種類の方法でテストコードを記述して，テストカバレッジとミューテーション解析の測定値をもとに，どの方法が最も良いか比較実験を行った．

その結果,「RQ1: 方法 1（ソロ 2 台）と方法 2（ペア 1 台），方法 3（ペア 2 台）のいずれの方法が，最も高い品質のテストコード記述を実現できるか？」という研究課題に対して，**概して，方法 3（ペア 2 台）を利用する場合にテストコードの品質が最も良く，方法 2（ペア 1 台）を利用する場合にテストコードの品質が最も悪かった**という結果が得られた.「RQ2: 方法 1（ソロ 2 台）と方法 2（ペア 1 台），方法 3（ペア 2 台）の 3 種類の方法に，対象とするプログラムによって向き不向きは存在するか？」という研究課題に対して，**取り組む課題の内容が難しく取り組む期間が短い場合は方法 1（ソロ 2 台）の方が良かった**という結果が得られた．また,「RQ3: 方法 1（ソロ 2 台）と方法 2（ペア 1 台），方法 3（ペア 2 台）のいずれの方法が，最も多くのペアに好まれるか？」という研究課題に対して，**テストコードの品質は最も悪かったのにも関わらず，方法 2（ペア 1 台）が最も多くのペアから好まれた**という結果が得られた．

今回の比較実験では，小規模なプログラムに対して 18 分という短期間でテストコード記述を行ったため，今後は，より規模の大きいプログラムに対して，より長い期間でテストコードを記述する実験を行う予定である．また，被験者の特徴や習熟性に関して偏りがあったかどうかを十分に確認ができなかったため，ペアプログラミングの有効性に影響があると分かっている要素について偏りが生じないようにペアを作り，その上で，ペアテスティングの有効性と有効な方法を検証するための実験を行っていきたい．その他に，ゲーミフィケーションを導入することで，ペアテスティングの有効性を高めるツールや手法を提案したい．

謝辞　本実験に協力して頂いた早稲田大学基幹理工学部情報理工学科の大学 4 年生，および，同大学院基幹理工学研究科情報理工学専攻の大学院生 36 名の学生に深く感謝します．

参考文献

[1] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[2] L. Williams, R.R. Kessler, W. Cunningham, and Ron Jeffries. Strengthening the case for pair programming. *Software, IEEE*, Vol. 17, No. 4, pp. 19–25, 2000.

[3] Hanna Hulkko and Pekka Abrahamsson. A multiple case study on the impact of pair programming on product quality. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pp. 495–504, New York, NY, USA, 2005. ACM.

[4] Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. The effects of pair-programming on performance in an introductory programming course. *SIGCSE Bull.*, Vol. 34, No. 1, pp. 38–42, February 2002.

[5] Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. Improving the cs1 experience with pair programming. *SIGCSE Bull.*, Vol. 35, No. 1, pp. 359–362, January 2003.

[6] Linda L. Werner, Brian Hanks, and Charlie McDowell. Pair-programming helps female computer science students. *J. Educ. Resour. Comput.*, Vol. 4, No. 1, March 2004.

[7] Charlie McDowell, Linda Werner, Heather E. Bullock, and Julian Fernald. Pair programming improves student retention, confidence, and program quality. *Commun. ACM*, Vol. 49, No. 8, pp. 90–95, August 2006.

[8] Erik Arisholm, Hans Gallis, Tore Dyba, and Dag I.K. Sjoberg. Evaluating pair programming with respect to system complexity and programmer expertise. *IEEE Trans. Softw. Eng.*, Vol. 33, No. 2, pp. 65–86, February 2007.

[9] Jo E. Hannay, Tore Dybå, Erik Arisholm, and Dag I. K. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Inf. Softw. Technol.*, Vol. 51, No. 7, pp. 1110–1122, July 2009.

[10] Norsaremah Salleh, Emilia Mendes, John Grundy, and Giles St. J Burch. An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pp. 577–586, New York, NY, USA, 2010. ACM.

[11] A. Sillitti, G. Succi, and J. Vlasenko. Understanding the impact of pair programming on developers attention: A case study on a large industrial experimentation. In *Software Engineering (ICSE), 2012 34th International Conference on*, pp. 1094–1101, 2012.

[12] A.S.M. Sajeev and Subhajit Datta. Introducing programmers to pair programming: A controlled experiment. In Hubert Baumeister and Barbara Weber, editors, *Agile Processes in Software Engineering and Extreme Programming*, Vol. 149 of *Lecture Notes in Business Information Processing*, pp. 31–45. Springer Berlin Heidelberg, 2013.

[13] Hong Zhu, Patrick A. V. Hall, and John H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, Vol. 29, No. 4, pp. 366–427, December 1997.

[14] Y.K. Malaiya, N. Li, J. Bieman, R. Karcich, and B. Skibbe. The relationship between test coverage and reliability. In *Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on*, pp. 186–195, 1994.

[15] L. Briand and D. Pfahl. Using simulation for assessing the real impact of test coverage on defect coverage. In *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on*, pp. 148–157, 1999.

[16] W.E. Howden. Weak mutation testing and completeness of test sets. *Software Engineering, IEEE Transactions on*, Vol. SE-8, No. 4, pp. 371–379, 1982.

[17] J.H. Andrews, L.C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? [software testing]. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pp. 402–411, 2005.

[18] John T. Nosek. The case for collaborative programming. *Commun. ACM*, Vol. 41, No. 3, pp. 105–108, March 1998.

[19] Jan Chong and Tom Hurlbutt. The social dynamics of pair programming. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pp. 354–363, Washington, DC, USA, 2007. IEEE Computer Society.

[20] D. Schuler and A. Zeller. Assessing oracle quality with checked coverage. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pp. 90–99, 2011.

[21] Michael Whalen, Gregory Gay, Dongjiang You, Mats P. E. Heimdahl, and Matt Staats. Observable modified condition/decision coverage. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pp. 102–111, Piscataway, NJ, USA, 2013. IEEE Press.