NAIST-IS-MT1051062

修士論文

ソースコードの読解方法とレビューアの経験が欠陥検出の効果と効率に与える影響の分析

田口 雅裕

2012年2月2日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

田口 雅裕

審査委員：

松本 健一 教授	（主指導教員）
飯田 元 教授	（副指導教員）
門田 暁人 准教授	（副指導教員）
森崎 修司 助教	（静岡大学）

ソースコードの読解方法とレビューアの経験が欠陥検出の効果と効率に与える影響の分析*

田口 雅裕

内容梗概

保守，派生開発で実施されるソースコードレビューにおいて，レビューアの経験や読解方法がレビューアの理解度とレビュー時間にどのような影響を与えるかを明らかにすることを目的として実験を実施した．実験では，65 名の被験者にペイントアプリケーションの Version1.0 である既存のソースコード，4 件の変更仕様，各変更仕様に対応する変更後のソースコードをレビューしてもらった．被験者はレビュー対象の閲覧方法として，紙 (印刷したソースコード) と計算機画面を選ぶことができる．また，各変更仕様のレビュー時間，変更後のソースコードが変更仕様を実現するものになっているかどうかとその判断理由を回答してもらい，個々の判断理由から被験者の理解度を 3 段階に分類した．レビューの重点読解箇所，Version1 の読解方針，対象ソースコードの閲覧方法の 3 種類を被験者の読解方法とし，読解方法と経験がレビュー時間と理解度に与える影響を調べたところ，豊富な経験によりレビュー時間が短くなること，変更にあわせて読解方法を選択することにより理解度が高くなることがわかった．

キーワード

コードレビュー; プログラム理解; 保守開発; エンピリカルソフトウェア工学;

* 奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文, NAIST-IS-MT1051062, 2012 年 2 月 2 日.

Analysis of the Effects of Reviewer's Experiences and Methods for Comprehending Source Code on Efficiency of Detecting Defect*

Masahiro Taguchi

Abstract

The goal of this study is to investigate the effect of reviewers' experience and comprehension approach on the time to review and comprehension level of source code review for software maintenance and evolution. We conducted an empirical study with software development practitioners as subjects. The subjects were asked to comprehend existing source code (Version 1.0) and review some modified source codes including Patch file format for corresponding to 4 change specifications. The subjects could use papers printed source code or PC screen for review. And then the subjects were asked also to record the time to review each patch and judge whether each patch meets the change specification to describe the reason for each patch judgments. Each reason for the judgment was categorized into one of three comprehension levels of the patch.

This thesis analyze the effects 5 kinds of experience and 3 method for review (comprehending policy for Version 1.0, media for showing souce codes and place of focus for review) on review times and comprehension level.

The results from the 65 industry practitioners who finished the code review indicated that the reviewers' experience reduced review times but did not help reviewers to comprehend in more detail. The results also indicated that the

* Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT1051062, February 2, 2012.

reviewers' comprehension approach increase the level of comprehension but did not reduce the review times.

Keywords:

code inspection; program comprehension; software maintenance; empirical software engineering;

関連発表論文

研究会・全国大会等

1. 田口 雅裕, 森崎 修司, 松本 健一, “ソースコード理解に求められる知識が理解時間に与える影響の実験的評価,” 平成 22 年度 情報処理学会関西支部支部大会 講演論文集, Vol.2010, September 2010.

国内会議

1. 田口 雅裕, 森崎 修司, 松本 健一, “ソースコードの差分のレビューにおけるレビューアの理解度の実験的評価,” ソフトウェア工学の基礎 XVIII, 日本ソフトウェア科学会 FOSE2011, pp.71-80, November 2011.

目 次

図 目 次

表 目 次

1. はじめに

近年のソフトウェアは顧客の需要を満たすために絶え間なく改変が加えられている．ソフトウェアリリース後に加えるソフトウェアの改変作業はソフトウェア保守開発と言われ [15]，ソフトウェア保守開発はソフトウェアライフサイクルにおいて 60% から 80% のコストを占めると言われている [5]．そのため，これまでにソフトウェア保守開発におけるコスト見積もりや，保守工程で発生した問題の要因分析などが盛んに研究されてきた [2, 3, 6]．Corbi [6] 等の研究ではソフトウェア保守開発にかかるコストの 60% は保守対象となるソフトウェアの構成，仕様や機能を理解することであるとも報告されており，ソフトウェア保守開発の中でも既存のバージョンを理解するためのプログラム読解やソフトウェア変更作業の効率化に関する研究が実施されてきた [7]．特に，プログラム読解はソフトウェア保守開発においても主要かつ高コストな活動であると言われており [23]，JIS96 で定義されている保守プロセスにおいてはプログラム読解はソフトウェアの変更時以外に，保守レビュー（保守開発時のソフトウェアレビュー）時においても実施される．

ソフトウェア保守開発では，機能追加や機能変更，機能削除そして機能修正等のソフトウェア改変活動を効率的に行うことが重要であると上記で述べたが，それと同時にソフトウェアの品質を維持するために，既存のソースコードに欠陥が混入しないようにするためにはソフトウェアレビューの効率化も重要である．レビューを保守開発工程に組み込むことによって開発工数の増加も考えられるが，過去の研究の中には [14, 25] ソースコードレビューを通じてソフトウェアの保守性に悪影響を与える欠陥や問題点を検出することはその後のソフトウェア保守開発を効率的に行うためにも重要性があると示されている．しかしながら，ソフトウェアレビューの代表的な支援方法であるチェックリストでも作成コストが大きいといった難点もあり，レビューの効率化については未だ多くの課題を残している．

ソフトウェアレビューの効率化のアプローチの一つとしてレビューアの割り当て最適化が挙げられる．レビューは人間によるソフトウェアの静的解析であるためレビューアの経験による個人差が大きいと言われている．しかしながら，レビューアの経験や読解法がレビュー結果にどのような影響を与えるのかは明らか

ではなく，仮に経験豊富なレビューアの方が効率的に正しい読解法でレビューできたとしても実務の現実を考慮すればそのような人材が常にレビューに参加することは困難である．そのような場合にもソフトウェアレビューを効率的に実施するためにはレビューアの開発経験や読解法によるレビュー品質への影響を把握し，その時に利用可能な人的リソースの中でレビューの品質を維持することが重要であると考えられる．さらに，近年のソフトウェアの機能多様化や大規模化を考えると，ソフトウェアフレームワークやコンポーネント，ライブラリが多く用いられるようになっているため，用いられている技術に関するレビューアの開発経験がレビューの効率にどのような影響を与えるかや，レビューアがどのようにソフトウェア理解を行うとレビューを効率的に完了できるのかを知ることが重要であると考えられる．ネットワークプロトコルに関するプログラミングに慣れたレビューアはソフトウェアのネットワーク通信部のコードレビューを効率的に行うことができるが，データベース関連のソースコードレビューでは同じようにはならないことが考えられる．しかしながら，ライブラリの利用経験のようなソフトウェア固有の経験によるレビュー効率への影響はこれまでのところ明らかになっておらず，レビューアとして選抜する人材を適切に選べないなどの問題が生じている．ソースコードレビューに関する先行研究では，ソースコードレビューで発見される欠陥の分類 [20, 25] やソースコード読解と読解時間の関係の分析 [13] などが研究されてきたがレビューアの開発経験や読解法がレビューの効率にもたらす影響は考慮されていなかった．また，これまでは欠陥を指摘できたかそうでないかといった評価が多く，被験者のレビュー成果が複数段階的に評価されてこなかった．そのためレビューで求められる品質とレビューアの開発経験や読解方法を踏まえた適切なレビュー実施法の提案がなされてこなかった．

本論文では被験者がレビューによって指摘した内容を 3 段階の理解度として評価し，レビューアの開発経験や読解法がレビュー時間や理解度にどのような影響を与えるかを明らかにすることを目的とする．レビューアの開発経験はライブラリの利用経験やレビュー対象のファイル形式の利用経験といったプログラミング経験のような一般的な経験だけではなく，細分化した開発経験を評価する．アプローチとして，ソフトウェア開発に携わる実務者を被験者としたレビューの実証

的実験を実施し以下の Research Question（以下 RQ）に回答する．

RQ1 レビューアの開発経験はレビュー時間・理解度にどのような影響を与えるか?

RQ2 レビューアの読解法はレビュー時間・理解度にどのような影響を与えるか？

これらの RQ に答えるため，実験において被験者には 4 つのソースコードパッチ（以下差分）をレビューしてもらい，レビュー結果として読解方針や各差分のレビュー時間を記録してもらう．また，被験者の開発経験の有無によってレビュー結果を比較するために被験者のソフトウェア開発経験をライブラリ利用経験やプログラミング経験などの 5 つの観点で問うアンケート形式で収集する．

以降，次章ではレビューアの経験や読解アプローチに関する関連研究，4 章では本研究の実験概要，5 章では実験結果について示し，6 章で結果について考察する．最後に 7 章で本論文をまとめる．

2. 関連研究

表 1，2，3 は本論文の関連研究をまとめたもので，それぞれの研究の RQ，実験の被験者，実験のタスク，分析に用いられた被験者の経験を示している．本論文では，関連研究の中から被験者の開発経験による影響を調査しているもの，被験者の読解法による影響を調査しているもの，その他の要因による影響を調査しているものに分類し，それぞれでいくつかの例を挙げる．関連研究と本論文との大まかな違いとして関連研究の多くは被験者として学生を採用していることが挙げられる．中にはソフトウェア開発に携わる実務者を被験者とした実験も行われているがその参加者数は最大でも 30 人となっており，本論文の実験の様に 65 人の実務者を対象としたものはない．また被験者の開発経験に関してもプログラミング経験の様な一般的な経験のみを評価に用いているものが多く，本論文のようにソフトウェアライブラリ利用経験などの多岐の観点で経験を考慮しているものは少ない．

2.1 開発経験の違いによる影響分析

表 1 に示す関連研究は被験者の経験がソースコードレビューやソースコード読解にどのような影響を与えるかどうかを分析したものである．被験者の開発経験の違いによる影響分析の研究では，初心者プログラマーや上級者プログラマー，手続き型プログラミング言語を利用するプログラマーやオブジェクト指向型プログラミング言語を利用するプログラマーといった経験の違いによって読解方針がどう違うか [4, 8, 9]，読解範囲がどう違うか [19, 28]，プログラミング計画がどう違うか [26] が研究されてきた．しかしながら，考慮する開発経験がプログラミング経験のみといった様に単純であり近年のフレームワーク化や大規模化といったソフトウェアの特徴を考えればより細かな開発経験による影響を考慮する必要があると考えられる．本論文では，GUI クラスライブラリ（Java/Swing）の利用経験やソースコードパッチの利用経験など複数の観点を考慮した分析を行う．以下に，表 1 中から分析内容の具体例を挙げる．

Burkhardt 等 [4] は初心者と上級者で読解方針がどのように異なるかを調査し

た．彼らは学生 21 人（初心者）と実務者 28 人（上級者）の読解方針が 3 つの観点，読解範囲，読解の優先順（トップダウンかボトムアップ），ソースコードの読み進め方（実行順に読むなど）においてどのように違うのかにについて研究した．読解方針の 3 つの観点の評価尺度はそれぞれ異なるファイルの読解回数，読解された上位レベル，下位レベルクラスの割合，コードの読解順を用いた．結果として，彼らは上級者は初級者に比べより多くの異なるファイルを読解し初級者は実行順にソースコードを読解する傾向にあることを報告した．彼らの研究はソフトウェア改変時のコード読解を対象としていないこと，また読解者の理解度を考慮していない点でも本研究とは異なる．

Soloway と Ehrlich [26] は一般的なコーディングの知識と一般的ではない形式で記述されたソースコードの読解について調べた．実験に参加した被験者は第一回目の Pascal プログラミングクラスに出席した 94 人学生を初級者プログラマーとし，少なくとも 3 回のプログラミングクラスを受講した 45 人の学生を上級者プログラマーとして取得単位数の違いで経験差を定義した．実験において被験者は一般的なコーディングスタイルで書かれたソースコードの穴埋め問題と，一般的でないコーディングスタイルで書かれたソースコードの穴埋め問題を回答するタスクが与えられ，Soloway 等は二種類のコーディングスタイルのコード読解中に経験の差がどのように現れるかを調査した．結果として，一般的でないコーディングの場合，穴埋め問題に回答するまでの時間が増加し正答率も減少することを報告した．また，これらの現象は特に上級者プログラマーに対してより顕著に現れたことから，一般的でないコーディングはソースコード読解の効率を著しく落とすことを指摘した．彼らの実験では小規模なソースコードを利用し，被験者の経験差は Pascal プログラミング経験のみを考慮したが，本論文ではフレームワーク利用経験など複数の経験も考慮する．また，Soloway 等の研究ではソースコードをどのように読んだかを考慮していないが本論文では経験と同時に読解法の影響も考慮する．

表 1 関連研究（開発経験の違いによる影響分析）

文献	Research Question	被験者 実務者	学生	タスク	被験者の経験尺度
Burkhardt [4]	実務者と学生で読解範囲や読解順に違いがあるか？	28	21	プログラム読解	実務者であるか学生であるかの違い
Corritore [9, 19]	オブジェクト指向プログラミング言語を主言語とする実務者は手続き型プログラミング言語を主言語とする被験者に比べ (1) トップダウンにプログラム読解を行うか，(2) 狭い読解範囲か？	30	0	メンテナンスタスク	主に利用するプログラミング言語の種類
Wiedenbeck [28]	初級者プログラマーは小規模なプログラムからどのような情報を抽出するか？	0	41	プログラム読解，読解したプログラムに関する筆記試験	プログラミング授業の単位数，主に利用するプログラミング言語
Soloway [26]	定石に従わないコーディングはプログラム読解を困難にするか？	0	139	プログラム読解，プログラムの穴埋め試験	プログラミング授業の単位数

2.2 読解法の違いによる影響分析

表 2 に示す関連研究は被験者がどのようにソースコードを読解したか，またその違いが結果にどのような影響を与えるかを調査しているものである．被験者の読解法の違いによる影響分析の研究では読解中にどのような行動をしているか [18, 24]，インスペクションメソッドの比較 [12, 13, 16, 22, 27] など効率的な読解法について研究されてきた．しかしながら，個々の被験者が持つ開発経験による影響を考慮できていないものが多く，被験者自身のスキルによって実験結果に違いが生じているのか，読解法によって違いが生じているのかが明らかではない．本論文では，被験者の開発経験と読解法を分離して考慮し，それぞれがレビュー時間と理解度にどのような影響を及ぼすのかを明らかにする．以下に，表 2 中から分析内容の具体例を挙げる．

Robillard 等 [24] は 5 人の学生を被験者としてソフトウェア変更タスクのためのソースコード読解中における開発者の活動について調査した．彼らは被験者のソースコード読解活動をビデオカメラで撮影し，記録した映像を解析することによって被験者の活動を分類した．また，タスクの完了時間の比較とタスクの達成度を 5 段階評価（バグを含んでいるか等）で分類し，達成度とタスクの完了時間を合わせて比較することでタスクを完了できた被験者とそうでない被験者を比較した．結果としてタスクを成功させそしてより短い時間でタスクを完了できた被験者はそうでかった被験者よりも体系的な手順で読解を進め，そうでない被験者は場当たり的にソースコード読解していると報告した．彼らの研究は被験者として学生を採用している点，被験者間の比較にプログラミング経験だけを用いている点で本論文とは異なる．

Karahasanović 等 [16] はソフトウェア保守開発におけるソースコード読解方針とその際の被験者が感じる主観的な読解難易度の関係について研究した．彼らの実験には 38 人の学生が参加し，体系立てられた方法でソースコード読解する場合と被験者が理解に必要な部分であると判断した部分だけ読解する場合とでメンテナンスタスクの完了時間とタスク成功率がどのように違うのかを評価した．結果として，体系立てられた方法で読解した被験者のほうがタスクを成功させやすいが，読解方法によって主観的な読解難度に違いが現れたことから，読解者の読

解方針の選択肢が少ないと読解範囲の網羅性がなくなることを指摘し，様々な読解法とその利点欠点を学ぶことが効率的ソースコード読解に重要であると提言した．彼らの研究では読解法に重点をおいているが各被験者の経験については考慮されていない点で本論文とは異なる．

表 2 関連研究（読解法の違いによる影響分析）

文献	Research Question	被験者 実務者	被験者 学生	タスク	被験者の経験尺度
Robillard [24]	修正タスクを成功する開発者にはどのような振る舞いが現れるか？	0	5	プログラム読解，プログラム変更	プログラミング経験年数
Ko [18]	(1) 開発者はどのようにコードの関係性を決定するか？(2) 開発者はどのような関係性を探すか？(3) 開発者はどのように関連情報を保持するか？	7	3	プログラム修正タスク	Java の上級者かそうでないかの違い
Dunsmore [12]	体系的な読解方法と場当たり的な読解方法ではどちらがより効率的か？	0	53	コードレビュー	Java，C 言語，SRS ドキュメントの利用経験，レビュー経験の有無
Karahasanović [16]	(1) 上級者は体系的な読解方法よりも場当たり的な読解方法を好むか？(2) 上級者が用いた読解方法と主観的な読解の困難さに関係はあるか？(3) 上級者はなぜ実験で用いた読解方法を利用したのか？	0	38	メンテナンスタスク	オブジェクト指向プログラミング言語又は Java プログラミングの経験，授業の単位数

2.3 その他要因の影響分析

表 3 に示す関連研究は前節，前々節で挙げた関連研究の他にソースコードレビューに関連したものである [1, 10, 11, 17, 20, 25].

Dunsmore 等 [11] は理解度と欠陥指摘数の関係について調査した．実験に参加した学生 47 人のレビュー実験結果と実験後に取得した被験者の主観的なソースコード理解度（10 段階評価）によって理解度と欠陥指摘数がどのような関係にあるのかを調べた．結果として，ソースコード理解度が高かった被験者の欠陥指摘数はそうでない被験者よりも多くなる傾向であると報告した．本論文では被験者の理解度の評価に欠陥指摘レポートの質を考慮するが Dunsmore 等は自己報告の理解度を用いている．また彼らは理解度と各被験者の開発経験との関係については分析していない点で異なる．

Mantyla と Lassenius [20] はコードレビュー時に検出される欠陥の種類について調査した．彼らは企業で実施されたコードレビューとアカデミックで実施されたコードレビューの結果を用いて分析し，被験者は 9 人の実務者と 23 人の学生だった．彼らは過去に定義された欠陥の種類を用いて企業，アカデミックそれぞれで実施されたコードレビューで発見された欠陥の種類を分類した．結果として，彼らはコードレビューによって検出される欠陥の 75% はソフトウェアの目に見える機能に影響するものではないことを報告した．しかしながら彼らの研究ではコードレビューの対象となったソフトウェアの特徴やどのような読解アプローチが取られたかについて言及されていない．

表 3 関連研究（その他要因の影響分析）

文献	Research Question	被験者 実務者	学生	タスク	被験者の経験尺度
Dunsmore [11]	プログラムの理解度（自己申告）は欠陥指摘数にどの様な影響を与えるか？	0	47	コードレビュー	-
Mantyla [20]	コードレビューで検出される欠陥の種類はどのようなものか？	9	23	コードレビュー	実務者であるか学生であるかの違い
Karahasanovic [17]	オブジェクト指向システムのメンテナンス時にどの様な読解の困難さを感じるか？	0	34	メンテナンスタスク	授業の単位数
Abbes [1]	スパゲッティコードやアンチパターンはコードの分かりやすさにどの様な影響を与えるか？	0	24	プログラム読解	Java，Eclipse の利用経験，ソフトウェア工学の知識の有無
Dunsmore [10]	どのような欠陥が容易に見つけることができないか？	0	47	コードレビュー	プログラミング経験年数

3. 保守レビュー

3.1 差分

ソフトウェアの保守開発では顧客の要求が変化したり，リリースしているソフトウェアに欠陥が発見されるなどして機能変更や，機能修正の作業が生じる．それらの活動は JIS X 0160 及び JIS X 0161:2008 で適応保守，是正保守などで定義されている．これらの作業では一般的には既存のソースコードに変更を加えることによって機能変更の要求に応えることが多く，差分とは変更を加える前のソースコードと変更を加えた後のソースコードの差を指す．また，変更前後の差とその周辺のソースコードを集約したものを差分ファイル（又はパッチファイル）と呼ぶ．差分の例については図 1 に示す通りである．

一般的にソフトウェアが大規模になるにつれて保守開発で変更を加える箇所は多くなり，ソースコードレベルでどのような変更が加えられたかの確認が難しくなる．また，変更を既存のソースコードに適用する（パッチを当てる，パッチを適用するとも言う）場合にも一つの修正でシステム全体のソースコードを入れ替えていては時間的なコストの増加にもつながる．そのような場合に差分ファイル

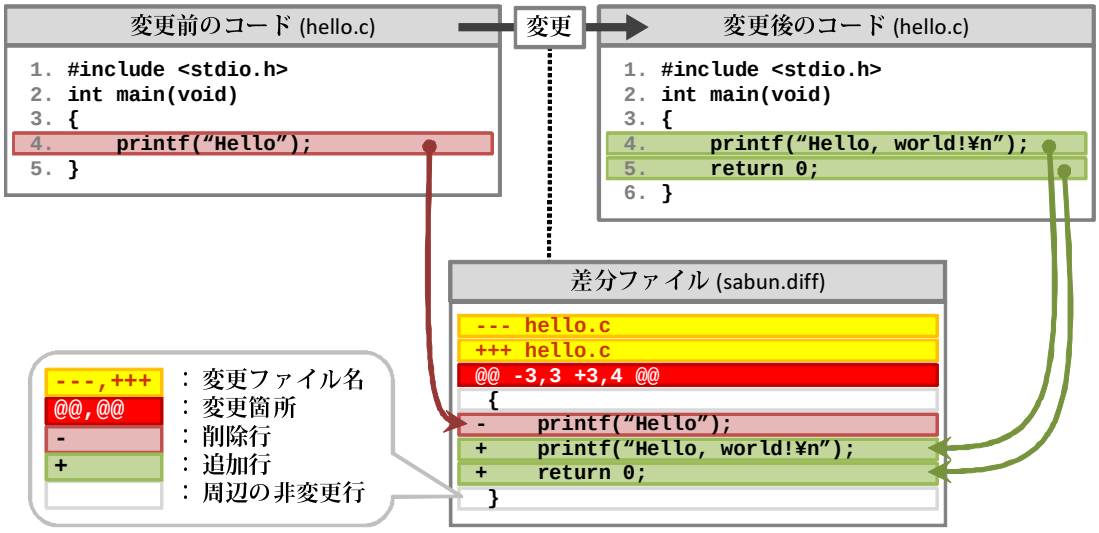


図 1 差分の例

形式を用いれば変更が加えられた箇所やその内容の確認も容易になり，変更適用の際には差分のみを入れ替えることで時間的なコストを削減することもできる．

3.2 保守開発時のソースコードレビュー

図 2 は JIS で定義された保守プロセスの概要図である．「プロセス開始の準備」で問題報告や修正依頼が起票され，実施が決定したもの一件ごとに対して「問題把握及び修正分析」,「修正の実施」,「保守レビュー及び受入れ」が原則として 1 回ずつ実施される．「移行」はソフトウェアを新しい環境の下で稼働させるために行う作業のことで,「ソフトウェア廃棄」は対象のソフトウェアが有用でなくなった場合に行う作業を指す．

保守レビューとはソフトウェアに加えられた修整を承認する権限を持つ組織の担当者を含めて共同で行う作業のことで，得られた成果物に対して目視で欠陥の混入やソフトウェアの完全性，保守性などの検査を行うプロセスを指す．レビューによって変更を適用しても良いかどうかの適用可否が判断され，レビュー対象物が求める品質の基準を満たしていればリリースが可能となる．レビューの一般的な利点は欠陥を検出することによるソフトウェア品質の確保という直接的なものだけでなく，レビュープロセスにおいて欠陥を早期に発見することで後工程で欠陥を発見するよりも手戻り工数が小さいとされ結果的に全体の工数削減につながるといった間接的なものが報告されている．レビューの中でも特にコーディング過程で得られたソースコードに対するレビューをソースコードレビューと呼ぶ．

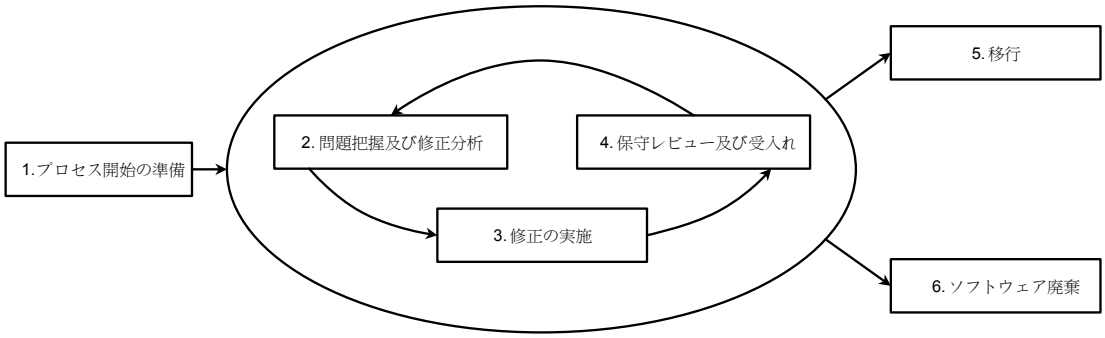


図 2 保守プロセス

保守開発時に実施するソースコードレビューは欠陥の検出以外にも，修正の対象となった箇所に確実に修正が施されているか，加えた修正によって変更要件を満たす事ができているか，修正が加えられたことによって既存の機能や性能，保守性を損なっていないかなどを加えて検査しなければならないといった特徴を持つ．特に保守開発の工数削減にもつながる保守性はテストでは発見が難しい検査項目であり，一般的にはソースコードレビューで検出されるとされている．

3.3 課題

2章で挙げたソースコードレビューの関連研究はどれもソフトウェア保守開発の効率化に大きな貢献を与えるものである．しかしながら，同章に挙げた文献の実験ではソフトウェア保守開発の結果に影響を与える要因として評価している因子の範囲が限定的である．例えば，読解法の評価ではタスクにかかった時間やタスクの成否を評価の判断材料にしているが，被験者の持つ経験や知識による影響を考慮できていない．被験者の開発経験の違いはタスクの成否を左右する可能性が考えられるが読解法の違いのみに着目している点が課題である．また，タスクの成否は機能実装の成否に着目するものが多く，タスクの質を段階的に評価しているものが少ない．しかしながら実務における有用性を考えれば成果の評価基準は複数設けられることが理想である（例えば，保守性を維持するための評価基準がなければ将来のメンテナンス作業にネガティブな影響を与える可能性がある [21].）．経験の評価では経験や知識によって読解方針に違いが現れることが分かっているが，それらがソフトウェア修正タスクやコードレビュー結果にどのような影響を与えるかまでは追求されていない．さらにこれまでの研究では主にプログラミング経験を被験者の経験として扱うことが主流であるが近年のソフトウェアの大規模化，フレームワーク化を考えれば，ソースコードライブラリの利用経験等，より多様な経験ががソースコードレビューに与える影響についても調査が必要である．被験者の保有する経験や知識によってレビュー結果に違いが現れるとすれば，読解法の違いのみに着目することや，開発経験の違いにだけ着目するだけではソースコードレビュー時に現れる傾向を正確に評価できているとは言えない．そのため，ソースコードレビューの効率化のためにはレビューアの経験や知識，読解法

がソースコードレビューに与える影響がどのように違うのかという知見を積む必要があると考えられる．また，その他の問題点として多くの実験では参加者が学生であり実務者を対象とした実験が少ないことや，参加者の数も少ないといった点で社会での再現性に乏しいことが挙げられる．

本論文では以上のような課題を考慮し，ソフトウェア開発に関する様々な経験と読解アプローチがソースコードレビューに与える影響をより多くの実務者による実験結果によって評価し現実的な知見の提案を試みる．

4. 実験

4.1 概要

レビューアの開発経験や読解法がレビュー時間や理解度にどのような影響を与えるのかを検証するためにソースコードレビュー実験を実施した．実験の設定としてソフトウェア保守開発を想定し，あるアプリケーションに変更が加えられた時のソースコードレビューを主題とし，パッチファイル（差分ファイル）を用いたパッチレビューを実施した．実験では，変更前のソースコード読解と変更後のソースコード読解を実施してもらい，変更後のソースコードが与えられた変更タスクの変更仕様を満たせているかどうかを被験者にレビューしてもらった．被験者にはレビューの記録としてレビュー時間及び，変更適用の可否とその判断理由を記述してもらった．また，レビューアの開発経験として主に利用しているプログラミング言語，Java/Swing ライブラリの利用経験，パッチファイル利用経験，過去に携わったことのあるソフトウェアの規模，プログラミング経験を，読解法としてパッチレビューで重点的に読んだ場所（重点読解箇所），レビューするソースコードの表示媒体（ソースコード表示媒体），バージョン 1.0 の読解方針を実験より被験者に記録してもらった．

本論文で実施したソースコードレビュー実験はワークショップの一部として実施された．IBM Rational 日本のメールマガジンの購読者にワークショップへの参加を呼びかけし，88 名のソフトウェア開発実務者が実験に参加した．

本論文では被験者の差分適用の可否とその判断理由を用いて間違った指摘をしているもの，仕様の確認がとれているもの，細かい指摘ができているものの 3 段階で分類しそれぞれを理解度 0，理解度 1，理解度 2 として定義した．また，本論文で設定した RQ に回答するため，RQ1，2 のそれぞれを以下に分割し，分割した RQ に沿って実験データを分析した．

RQ1 RQ1.1 レビューするプログラミング言語の種類（オブジェクト指向プログラミング言語又は手続き型プログラミング言語等）が普段から利用している言語と同じかどうかによってレビュー時間や理解度に違いが現れる．

RQ1.2 レビューするソースコードに頻繁に用いられるライブラリの利用経験の有無によってレビュー時間や理解度に違いが現れる．

RQ1.3 レビューするファイル形式の利用経験の有無によってレビュー時間や理解度に違いが現れる．

RQ1.4 過去で開発に携わったことのあるソフトウェアの規模の違いによってレビュー時間や理解度に違いが現れる．

RQ1.5 プログラミング経験年数の違いによってレビュー時間や理解度に違いが現れる．

RQ2 RQ2.1 重点読解箇所の違いによってレビュー時間や理解度に違いが現れる．

RQ2.2 ソースコード表示媒体の違いによってレビュー時間や理解度に違いが現れる．

RQ2.3 バージョン 1.0 読解方針の違いによってレビュー時間や理解度に違いが現れる．

4.2 資料

4.2.1 ペイントアプリケーション（バージョン 1.0）

本実験でレビュー対象となるアプリケーションである．Java アプレット形式で実装されており，実験中は PC での利用を許可した．本アプリケーションは線描画，色選択，線消去や線で囲まれた領域の塗りつぶしなどが基本機能として備わっている．図 3 にペイントアプリケーションのスクリーンショットを示す．

4.2.2 バージョン 1.0 ソースコード

ペイントアプリケーションのソースコードである．変更を加える前のソースコードを指し，プログラミング言語は Java 言語によって書かれている．ソースコードの規模は 10 クラスから構成され，総行数は 1053 行である．ペイントアプリケーションの設計はワークショップの限られた時間の中で被験者がレビューを完了できるように十分に小さくする必要があった．

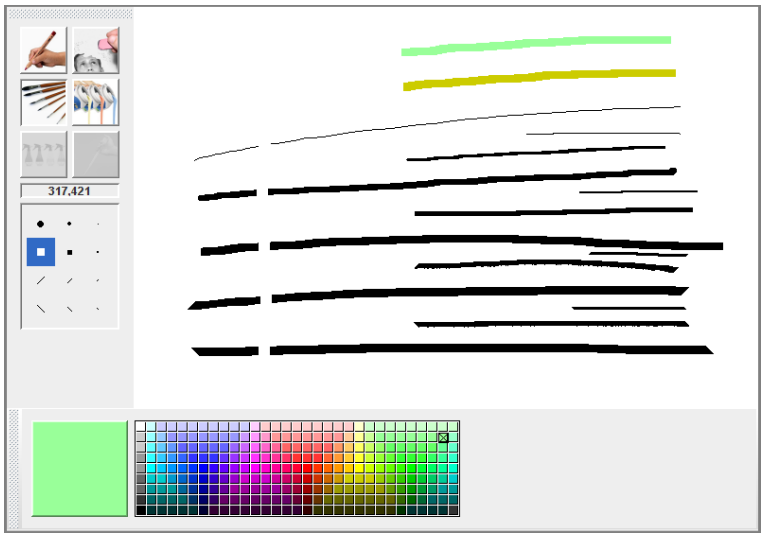


図 3　ペイントアプリケーションのスクリーンショット

4.2.3　変更タスクドキュメント

バージョン 1.0 に加える変更内容を示したものである．本実験では 4 つの変更タスクを用意した．本資料には変更内容，変更前の挙動，変更理由が自然言語で記されており，変更前後のアプリケーションのスクリーンショットも掲載している．また，それぞれの変更に対応するソースコードパッチファイル（差分ファイル）が掲載されている．図 4 に変更タスクドキュメントの一例を示す．

4.2.4　パッチファイル（差分ファイル）

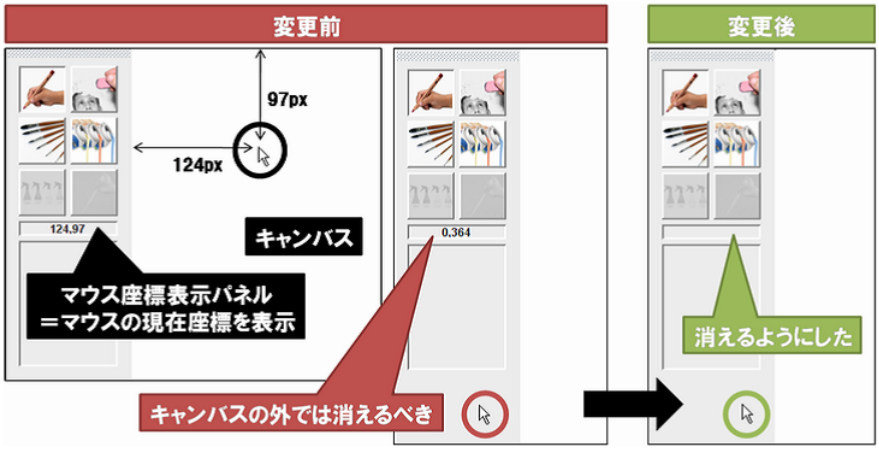
パッチファイルはソースコードの改変時において変更前と変更後のソースコードの差分を集約したもので差分ファイルとも言う．差分ファイルを見ることで変更されたファイル名，変更箇所，削除された行と追加された行及びその周辺の非変更行が一覧できる．図 5 に差分ファイルの例を示す．

差分 **01**

「マウス座標表示パネル」の修正

変更

変更内容: マウスカーソルがキャンバス(絵の描画領域)の外に出たら「マウス座標表示パネル」に表示されているマウスの画面上の座標を消す。
変更前: マウスカーソルがキャンバス(絵の描画領域)の範囲外に移動した場合でも、マウスカーソルがキャンバス外に移動する直前の座標が表示されたままになる。
変更理由: マウスがキャンバス内にないことを明示的に示したいため



差分情報

パッチファイル（Unified diffs）

```
--- C:/Documents and Settings/yuichiro-y/workspace/reviewpaint/trunk/ReviewPaintMain.java   Mon Jun 01 05:12:12 2009
+++ C:/Documents and Settings/yuichiro-y/workspace/reviewpaint/branches/modify_cursor_position/ReviewPaintMain.java F
    01:09:36 2009
@@ -158,6 +158,12 @@

             jPanelCanvasPaper = new CanvasPanel(pt_pencil); // キャンバスクラスの生成、デフォルトのペイントツールの指定

+            // キャンバスからマウスが離れた時のイベントリスナ
+            jPanelCanvasPaper.addMouseListener(new java.awt.event.MouseAdapter() {
+                public void mouseExited(java.awt.event.MouseEvent e) {
+                    jLabelCursorPosition.setText(""); // 座標表示をオフにする
+                }
+            });
             // マウス移動時・マウスドラッグ時のイベントリスナ
             jPanelCanvasPaper.addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
                 public void mouseMoved(java.awt.event.MouseEvent e) {
```

変更後のファイル

変更の種類	変更後のファイル	変更前後の比較
修正	/branches/modify_cursor_position/ReviewPaintMain.java	Side-by-side diffs

リンク先は Google code リポジトリ上のURLです

図 4 変更タスクドキュメント

4.2.5 変更前後のソースコード

パッチファイルとは別に変更前と変更後のソースコードを二列で並べたものである．修正箇所にはハイライトが施され，追加又は削除の違いによって色分けが

```
--- CanvasPanel.java    Mon Jun 15 14:36:56 2009
+++ CanvasPanel.java    Mon Jun 15 22:47:01 2009
@@ -80,6 +80,9 @@
             if (img == null) {
                 // 自身のサイズでイメージデータを初期化する
                 img = (BufferedImage) createImage(getSize().width, getSize().height);
+
+                // キャンバスに新しいイメージを描写する
+                ((Graphics2D) g).drawImage(img, 0, 0, this);
             }
             // 初期化済みの場合
             else {
@@ -102,8 +105,8 @@
     // キャンバスを白紙にする
     // ----------------------------------------------------
     public void Clear() {
-        // 自身のサイズでイメージデータを初期化する
-        img = (BufferedImage) createImage(getSize().width, getSize().height);
+        // 初期化
+        img = null;
         // 描写処理へ
         repaint();
     }
```

図 5　差分ファイルの例

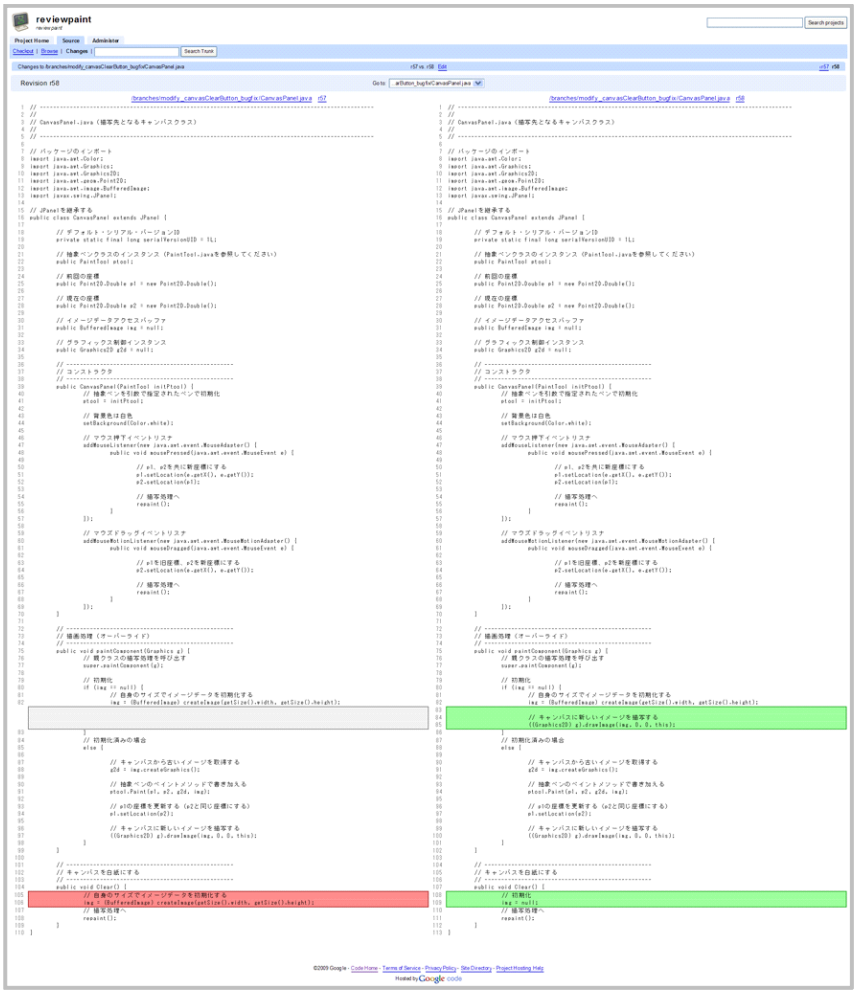


図 6　変更前後のソースコード

なされる．本資料は変更が加えられたファイルのソースコードのみを掲載している．図 6 に一例を示す．

4.3 変更タスク

本実験では 4 種類の変更タスクを用意した．表 4 にその概要を示す．用意した差分は GUI コンポーネントに変更を加える差分 01，02，03 と GUI コンポーネント以外に変更を加える差分 04 から構成される．また，それ以外の特徴として変更箇所数が挙げられる．変更箇所数とは追加，削除又は修正が加えられる行の集合（ソースコード片）の数を示す．つまり図 5 を例にとると削除される行が 1 行，追加される行が 2 行の計 3 行の変更であるが変更箇所が一箇所に集中しているため変更箇所数は 1 となる．一方で図 6 の様に追加するソースコードが二箇所に分かれている場合には変更箇所数が 2 となる．表中の説明は差分を適用する前と適用した後のペイントアプリケーションがどのように変わるのかを示している．

表 4 変更タスクの概要

ID	タイトル	GUI	変更箇所数	説明
01	マウス座標表示パネルの修正	変更あり	1	(変更前) マウスポインタがキャンバス領域外を指していてもマウスポインタの座標が常にパネルに表示される．(変更後) マウスポインタがキャンバス領域内を指しているときのみ座標をパネルに表示する．
02	カラーチューザの変更	変更あり	1	(変更前) カラーチューザは 300 色から構成され，ユーザはその中から好きな色を選択できる．(変更後) カラーチューザはグレースケール（9 色）のみから構成され，ユーザはそれ以外の色を選択できない．
03	ブラシ風描画機能の削除	変更あり	10	(変更前) ペイントアプリケーションはブラシ風描画機能を備えており，ユーザはツールパネル上に配置されたボタンを押下することでこれを利用できる．(変更後) ペイントアプリケーションからブラシ風描画機能が削除され，ツールパネル上のボタンも同時に削除される．
04	クラス名の変更	変更なし	11	PaintTool クラスを PenTool クラスに名称変更する．

4.4 手順

図 7 に本実験で被験者が行う実験手順を示す．本実験は図 7 に示すように 3 つの手順から構成された．

まず初めに，被験者にはバージョン 1.0 読解としてバージョン 1.0 の仕様説明やソースコードを読んでもらったり，実際にアプリケーションを実行してもらうことでバージョン 1.0 のアプリケーションがどういうものなのかを理解してもらった．また，この時被験者には変更タスクドキュメントが渡されたが，バージョン 1.0 の理解が終了するまで被験者には内容を見ないようにしてもらった．バージョン 1.0 読解の終了基準は将来に機能変更が起きることを想定しながら読解し被験者自身でそれが十分であると判断できるまでとした．バージョン 1.0 読解が終了した後，被験者にはバージョン 1.0 をどのように読解したのかについて記述してもらった．

次に，被験者には差分レビューとして変更仕様を理解してもらい，その変更に対応する差分を読解してもらい，その後で差分を適用しても良いかどうかを差分適応可否として判断してもらった．また，差分適応可否の判断理由を記述してもらった．実験に用いた変更タスクを表 4 に示す．表中の変更箇所数はバージョン 1.0 から変更されるコード片の個数を表している．差分レビューの終了の基準は被験者自身の判断に一任し，各差分レビューが終了した後，被験者には各変更タスクの理解に要した時間など以下の項目について記録してもらった．用意した差分は 4 つのうちどれからレビューしても良いものとした．

- 差分レビュー時間
- 差分適用の可否（二択形式）
- 差分適用可否の判断理由

最後に，被験者自身のソフトウェア開発経験のアンケートに回答してもらった．アンケート項目は開発経験に関するもの，読解アプローチに関するもので計測方法は記述式，選択形式を含むもので以下に実際のアンケート項目を示す．

- 経験に関するアンケート項目

- 主に利用しているプログラミング言語（自由記述）(以下，主言語)
- Java/Swing ライブラリの利用経験（経験あり，経験なしの二択）
- パッチファイル利用経験（パッチを用いた開発経験がある，パッチがどのようなものか知っているが開発に用いた経験はない，パッチがどのようなものか知らず開発に用いた経験もないの三択）
- 携わったソフトウェアの規模（10KLOC以下，10Kから1MLOC，1MLOC以上の三択）
- プログラミング経験（経験年数を記述）

- 読解法に関するアンケート項目
  - 重点読解箇所（パッチ自体を中心に読んだ，パッチとパッチ適用後のコードをまんべんなく読んだ，パッチが適用されたソースコードの周辺を中心に読んだの三択）
  - ソースコード表示媒体（PC，紙，PC と紙の三択）
  - バージョン 1.0 の読解方針（自由記述）

その他，実験中の制限として被験者はレビュー中のにパッチを適用してアプリケーションを実行することはできない，他の被験者と相談してはいけない，ワークショップの全体の実施時間は三時間までを設けた．また，被験者には時間内に全ての変更タスクをレビューできなくても良いと伝え，変更タスクのレビュー順にも制限を設けていない．実験中は専門書や Java API を参考にすることや，インターネットを用いた情報検索も可能とした．

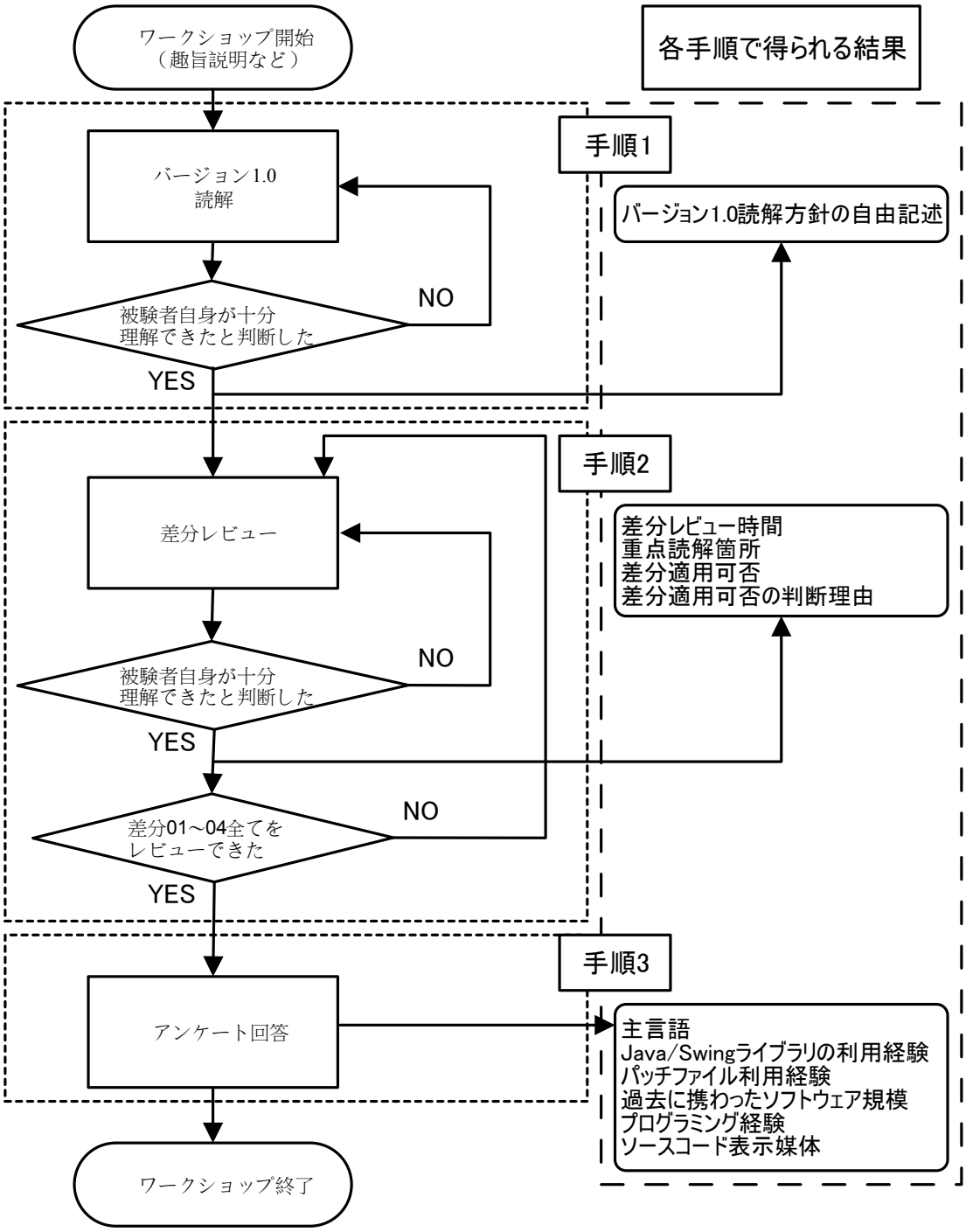


図 7 実験手順と各手順で得られる結果

5. 結果

5.1 バージョン 1.0 の読解方針分類

バージョン 1.0 読解方針の記述から二つの読解方針に被験者の読解方針を目視によって分類した．以下に読解方針の分類と分類基準について示す．

- 局所優先理解：18 名
  読解方針の記述に詳細に読んだことを示す記述があれば局所優先理解に分類される．具体例として，“ソースコードを上から下に順に読んだ”，“一行ずつ読んだ”，“プログラムの実行順に読んだ”，“仕様の実装方法まで理解した” などがこの読解方針が含まれる．

- 全体優先理解：42 名
  読解方針の記述に一行ずつ読まずにざっと目を通したことやクラスの依存関係などの抽象的な情報に注意を払った記述があれば全体優先理解に分類される．具体例として，“クラス図の依存関係を見た”，“コメントを中心に読んだ”，“実装方法”，“実装方法を気にせず，どのような機能かに注意を払った”，“全体構造の把握に専念した” などがこの読解方針に含まれる．

5.2 理解度分類

パッチ適用の可否とその判断理由を目視によって評価することで被験者の各差分理解度を三段階に分類した．以下に理解度の分類と分類基準について示す．

- 理解度 0
  差分適用可否と判断理由に矛盾があるものはこれに分類される．

- 理解度 1
  潜在的な問題，稀に起こる例外的事象を除き，通常利用の範囲内で差分適用可否に妥当な判断ができているものはこれに分類される．

- 理解度 2 理解度 1 に加え，潜在的な問題，稀に起こる例外事象を含め，詳細に確認した上での指摘が判断理由に含まれているものはこれに分類される．具体例として，“ドラッグ中には仕様を満たさない,” “マジックナンバーとなっている,” “不必要な変数が含まれている,” “不適切なコメントが存在する” などの指摘があるものを理解度 2 とした．

5.3 被験者の開発経験の違いによるレビュー時間と理解度の比較 (RQ1)

本節では，被験者の開発経験の違いによってレビュー時間と理解度に差が現れるかを検証した結果を示す．

図 8，10，12，14，16 は被験者の開発経験の違いによる被験者分類ごとのレビュー時間分布の比較を示している．これらの図中の縦軸はレビュー時間を表し，横軸は変更タスクの差分を表している．箱の下辺と上辺は各差分レビュー時間分布の下位四分位点と上限四分位点に対応している．箱中で横向きになっている黒線はレビュー時間分布の中央値を表すもので，下側及び上側の髭の末端はレビュー時間分布の最小値，最大値を表している．図上に並んだ数字は各経験の分類に該当した被験者の数を表している．

図 9，11，13，15，17 は経験の違いによる被験者分類ごとの理解度分布の比較を示している．これらの図中の縦軸は理解度分布の割合を表し，横軸は変更タスクの差分及び，各被験者分類を表している．図上に並んだ数字は理解度分類によって分類された各理解度の被験者数の分布を表している．

表 5，6，7，8，9 はレビュー時間，理解度それぞれの比較について統計的検定量を計算した結果を示している．レビュー時間の比較についてはクラスカル・ウォリス検定で全郡間の統計量を計算し，有意水準 5% を得られたものについては個別にマンホイットニーの U 検定により統計量を計算した．理解度の比較についてはフィッシャーの正確確率検定を用いて統計量を計算した．有意水準 5% 以下のものには*を 1% 以下のものには**を数値横に付加している．

5.3.1 主言語による比較 (RQ1.1)

図 8 は主に利用するプログラミング言語の違いによってレビュー間分布を比較したものである．全ての差分において，オブジェクト指向型プログラミング言語を主に利用している被験者のレビュー時間中央値は手続き型プログラミング言語を主に利用している被験者よりも小さくなった．

図 9 は主に利用するプログラミング言語の違いによって理解度分布を比較したものである．差分 01，02 では手続き型プログラミング言語を主言語とする被験者の方がオブジェクト指向プログラミング言語を主言語とする被験者よりも理解度 2 の割合が大きくなった．一方で，差分 03，04 ではオブジェクト指向プログラミング言語を主言語とする被験者のほうが手続き型プログラミング言語を主言語とする被験者よりも理解度 2 の割合が大きくなった．

表 5 は主言語による被験者分類毎のレビュー時間，理解度の分布について統計的検定量を計算した結果である．レビュー時間の分布はクラスカル・ウォリス検定によると全ての差分において統計的有意差は見られなかった．また，理解度分布についてもフィッシャーの正確確率検定によると全ての差分において独立性は認められなかった．

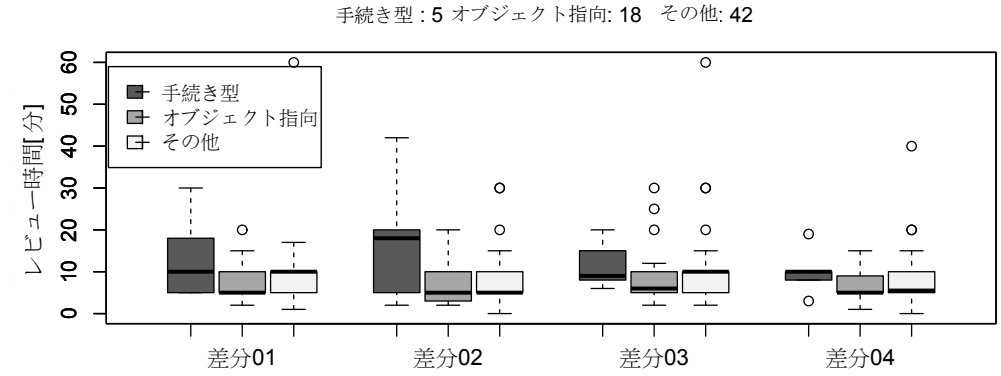


図 8　主言語によるレビュー時間比較

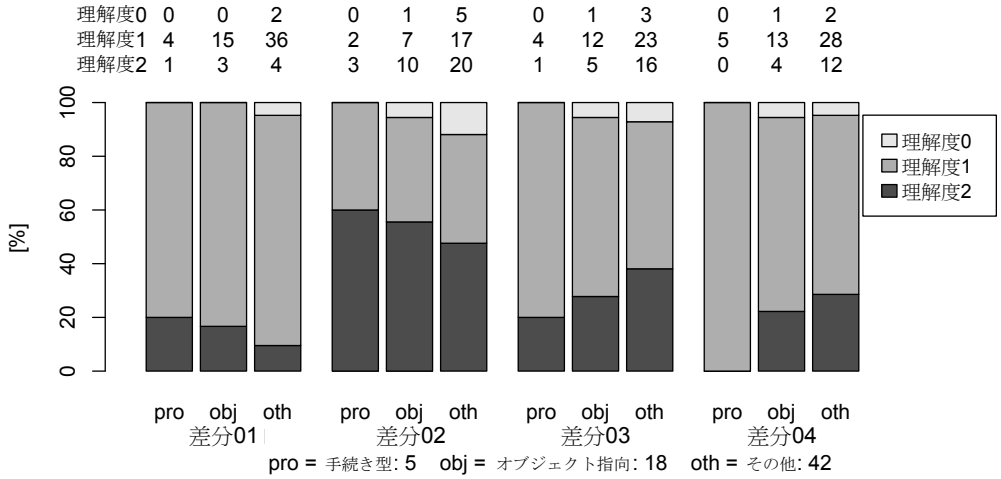


図 9　主言語による理解度比較

表 5　主言語によるレビュー時間・理解度の有意差検定結果

検定対象		p 値			
		差分 01	差分 02	差分 03	差分 04
レビュー時間	クラスカル・ウォリス検定	0.411	0.320	0.539	0.387
理解度		0.673	0.953	0.873	0.690

5.3.2 Java/Swing ライブラリ利用経験による比較 (RQ1.2)

図 10 は Java/Swing ライブラリの利用経験によってレビュー時間分布を比較したものである．Java/Swing の API が用いられる差分 01，02，03 では Java/Swing ライブラリの利用経験がある被験者のほうが利用経験がない被験者よりもレビュー時間中央値が小さくなった．しかしながら，Java/Swing に関係のない差分 04 では Java/Swing ライブラリの利用経験がある被験者よりも利用経験がない被験者のレビュー時間中央値が小さくなった．

図 11 は Java/Swing ライブラリの利用経験によって理解度分布を比較したものである．差分 03 においては，Java/Swing ライブラリ利用経験者の方が理解度 2 の割合が僅かに大きくなったが，差分 01，02，04 においては Java/Swing ライブラリの利用経験がない被験者のほうが理解度 2 の割合が大きくなった．

表 6 は Java/Swing ライブラリ利用経験による被験者分類毎のレビュー時間，理解度の分布について統計的検定量を計算した結果である．レビュー時間の分布はマンホイットニーの U 検定によると差分 03 以外には統計的有意差は見られなかった．一方，理解度分布についてはフィッシャーの正確確率検定によると全ての差分において独立性は認められなかった．

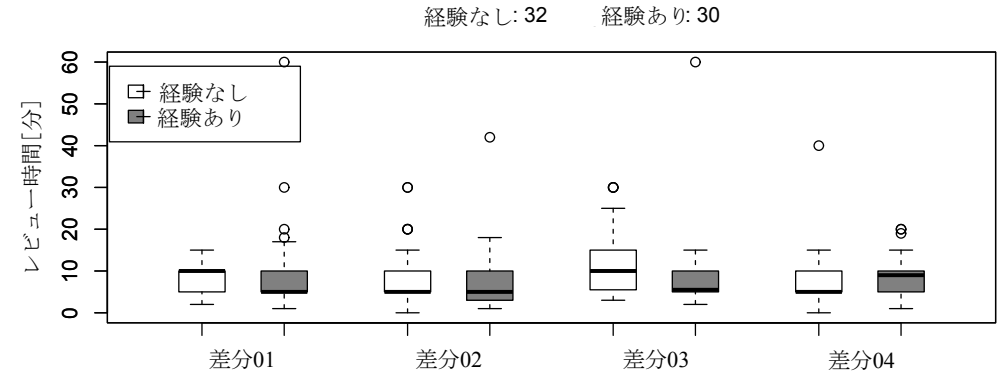


図 10 Java/Swing ライブラリ利用経験によるレビュー時間比較

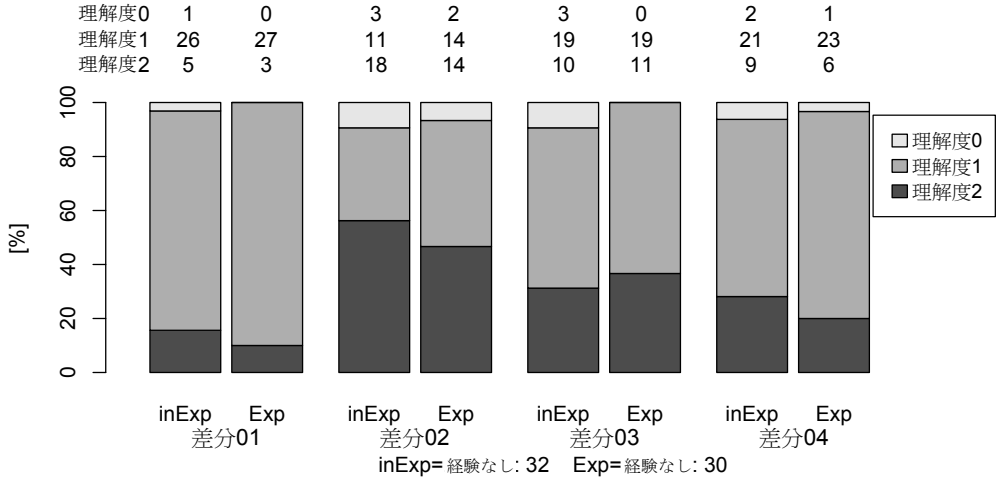


図 11 Java/Swing ライブラリ利用経験による理解度比較

表 6 Java/Swing ライブラリ利用経験によるレビュー時間・理解度の有意差検定結果

検定対象		p 値			
		差分 01	差分 02	差分 03	差分 04
レビュー時間	経験なし V.S. 経験あり	0.352	0.459	0.047 *	0.197
理解度		0.708	0.630	0.342	0.592

5.3.3 パッチファイル利用経験による比較 (RQ1.3)

図 12 はパッチファイル利用経験によってレビュー時間分布を比較したものである．全ての差分においてパッチ利用経験がある，もしくはパッチがどのようなものかを知っている被験者のレビュー時間中央値がそうでない被験者のレビュー時間中央値よりも小さくなった．特に変更箇所数の多い左軍 03，04 では，パッチ利用経験がある被験者のレビュー時間中央値がパッチがどのようなものかを知っているだけの被験者よりも小さくなった．

図 13 はパッチファイル利用経験によって理解度分布を比較したものである．各差分を通じての一貫した傾向は現れず，差分 01，02，04 ではパッチファイルを用いた開発経験はないがパッチファイルに関する知識を持っている被験者や，利用経験も知識もない被験者でも理解度 2 の割合が大きくなった．

表 7 はパッチファイル利用経験による被験者分類毎のレビュー時間，理解度の分布について統計的検定量を計算した結果である．レビュー時間の分布はマンホイットニーの U 検定によると差分 03 以外には統計的有意差は見られなかった．差分 03 についてマンホイットニーの U 検定で各群の統計量を計算した結果，経験なしと知識のみ，経験なしと利用経験ありの群間に有意水準 5% で有意差があった．一方，理解度分布についてはフィッシャーの正確確率検定によると全ての差分において独立性は認められなかった．

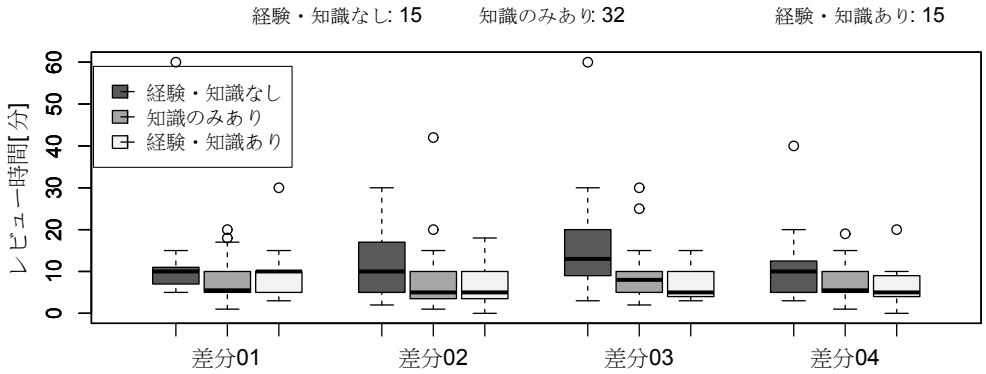


図 12 パッチファイル利用経験によるレビュー時間比較

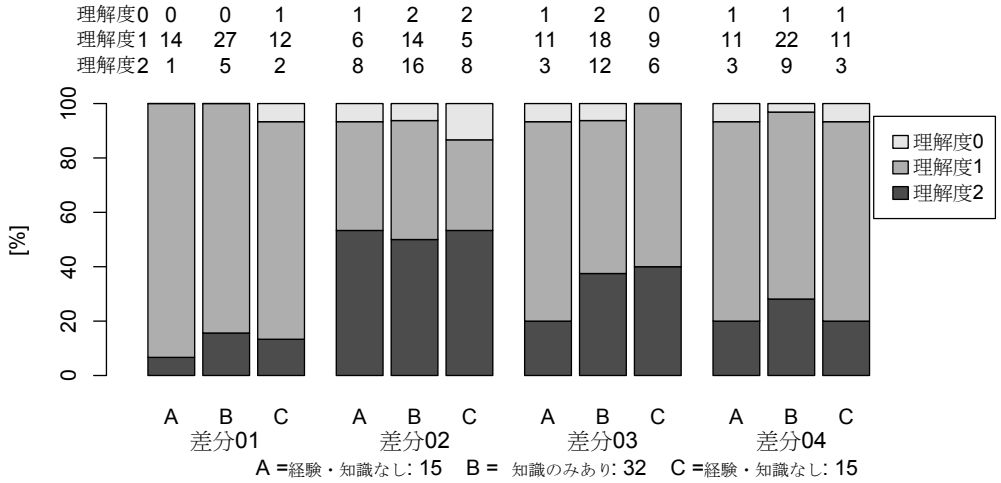


図 13 パッチファイル利用経験による理解度比較

表 7 パッチファイル利用経験によるレビュー時間・理解度の有意差検定結果

検定対象		p 値			
		差分 01	差分 02	差分 03	差分 04
レビュー時間	クラスカル・ウォリス検定	0.185	0.203	0.014 *	0.221
	経験なし V.S. 知識のみ	/	/	0.019 *	/
	経験なし V.S. 利用経験あり	/	/	0.009 **	/
	知識のみ V.S. 利用経験あり	/	/	0.294	/
理解度		0.560	0.916	0.683	0.890

5.3.4 携わったソフトウェアの規模による比較 (RQ1.4)

図 14 は過去に携わったことのあるソフトウェアの規模によってレビュー時間分布を比較したものである．全ての差分において，10KLOC 以下のソフトウェア開発経験しかない被験者よりもそれ以上の規模のソフトウェア開発の経験がある被験者のほうがレビュー時間中央値は小さくなった．特に差分 02，04 では，過去に携わったことのあるソフトウェアの規模が大きい被験者ほどレビュー時間の中央値が小さくなった．

図 15 は過去に携わったことのあるソフトウェアの規模によって理解度分布を比較したものである．差分 01，04 では 1MLOC 以上のソフトウェア開発経験がある被験者が理解度 2 の割合が最も大きくなったが，差分 02，03 では 10KLOC 以下，10K-1MLOC までのソフトウェア開発経験がある被験者のほうが理解度 2 の割合が大きくなった．

表 8 は携わったソフトウェアの規模による被験者分類毎のレビュー時間，理解度の分布について統計的検定量を計算した結果である．レビュー時間の分布はクラスカル・ウォリス検定によると差分 03 以外には統計的有意差が見られた．差分 01，02，03 についてマンホイットニーの U 検定で各群の統計量を計算した結果，すべての差分において 10KLOC 以下と 10K-1MLOC，10KLOC 以下と 1MLOC 以上の群間に有意水準 5% で有意差があった．一方，理解度分布についてはフィッシャーの正確確率検定によると全ての差分において独立性は認められなかった．

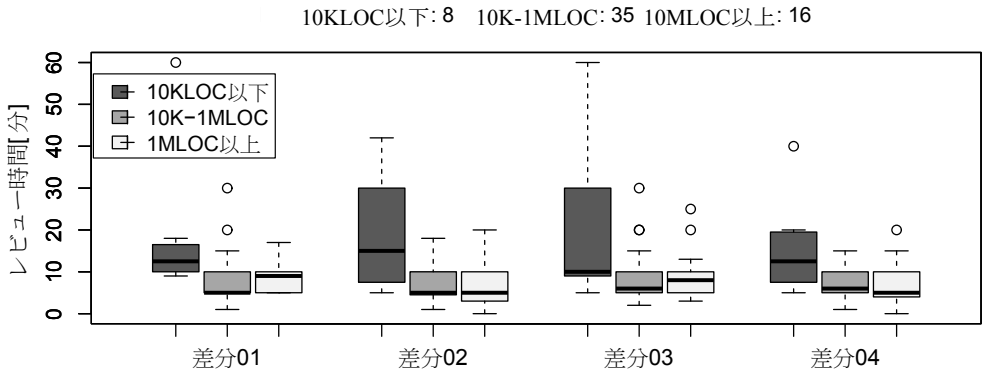


図 14　携わったソフトウェアの規模によるレビュー時間比較

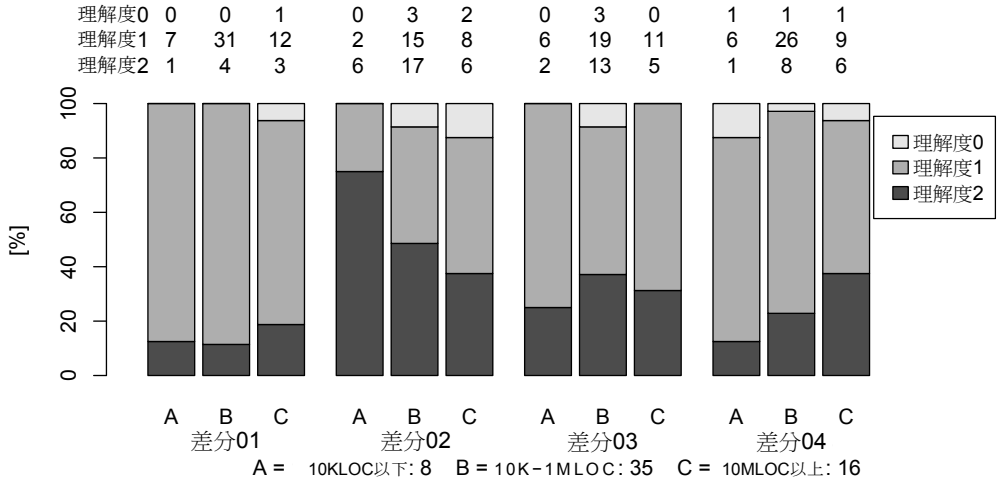


図 15　携わったソフトウェアの規模による理解度比較

表 8　携わったソフトウェアの規模によるレビュー時間・理解度の有意差検定結果

検定対象		p 値			
		差分 01	差分 02	差分 03	差分 04
レビュー時間	クラスカル・ウォリス検定	0.002 **	0.018 *	0.130	0.042 *
	10KLOC 以下 V.S. 10K-1MLOC	0.002 **	0.006 **	/	0.016 *
	10KLOC 以下 V.S. 1MLOC 以上	0.006 **	0.019 *	/	0.035 *
	10K-1MLOC V.S. 1MLOC 以上	0.056	0.778	/	0.661
理解度		0.519	0.587	0.774	0.396

5.3.5 プログラミング経験による比較 (RQ1.5)

図 16 はプログラミング経験によってレビュー時間分布を比較したものである．全ての差分において，5 年以上のプログラミング経験を持つ被験者のほうがそうでない被験者よりもレビュー時間中央値が小さくなった．特に差分 03,04 においてはその傾向が顕著に表れた．

図 17 はプログラミング経験によって理解度分布を比較したものである．差分 01，03 ではプログラミング経験が 5 年以上の被験者のほうが理解度 2 の割合が大きくなった．一方で差分 02，04 ではプログラミング経験が 5 年以下の被験者のほうが理解度 2 の割合が大きくなった．

表 9 はプログラミング経験による被験者分類毎のレビュー時間，理解度の分布について統計的検定量を計算した結果である．レビュー時間の分布はマンホイットニーの U 検定によると全ての差分において統計的有意差が見られなかった．また，理解度分布についてもフィッシャーの正確確率検定によると全ての差分において独立性は認められなかった．

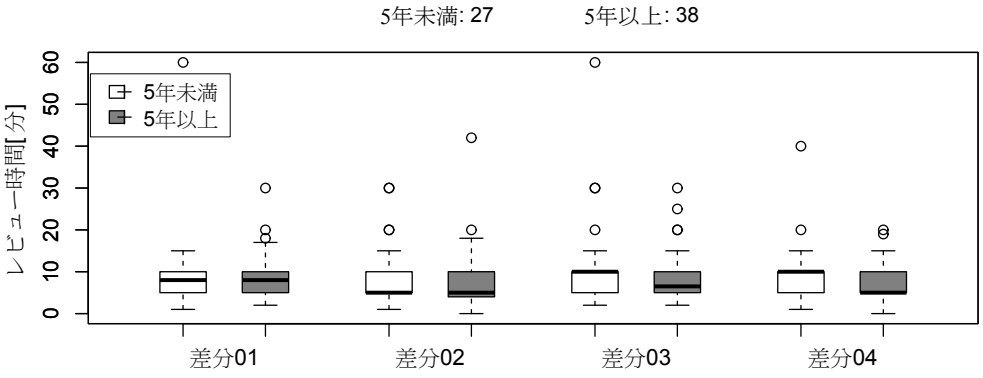


図 16 プログラミング経験によるレビュー時間比較

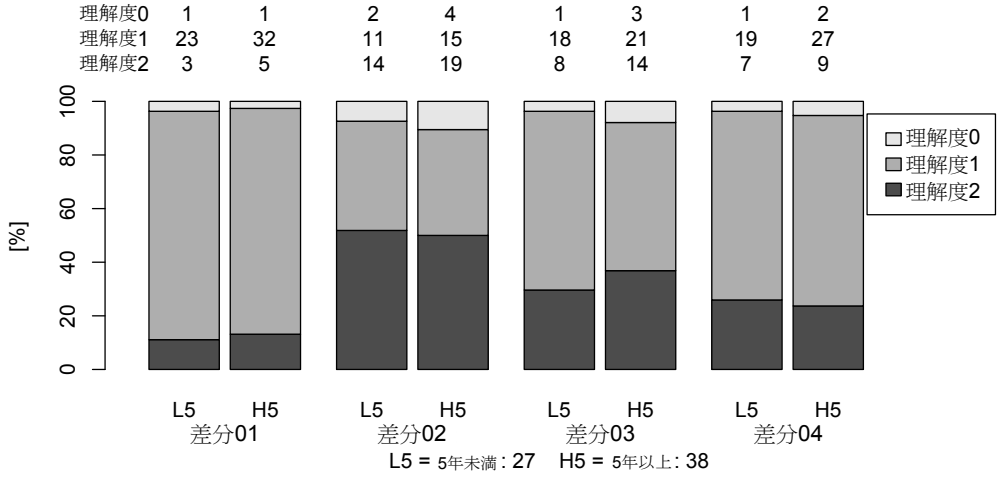


図 17 プログラミング経験による理解度比較

表 9 プログラミング経験によるレビュー時間・理解度の有意差検定結果

検定対象		p 値			
		差分 01	差分 02	差分 03	差分 04
レビュー時間	5 年未満 V.S. 5 年以上	0.848	0.492	0.529	0.314
理解度		1	1	0.649	1

5.4 被験者の読解法の違いによるレビュー時間と理解度の比較 (RQ2)

本節では，被験者の読解法の違いによってレビュー時間と理解度に差が現れるかを検証した結果を示す．

図 18，20，22 は読解アプローチの違いによる被験者分類ごとのレビュー時間分布の比較を示している．図 19，21，23 は読解アプローチの違いによる被験者分類ごとの理解度分布の比較を示している．また，表 10，11，12 はレビュー時間，理解度それぞれの比較について統計的検定量を計算した結果を示している．これらの図表の見方は前節で示したレビュー時間分布の比較図及び，理解度分布の比較図と同じである．

5.4.1 重点読解箇所による比較 (RQ2.1)

図 18 は差分レビューで中心に読んだ場所によってレビュー時間分布を比較したものである．短い時間でレビューを終えた読解箇所は差分によって異なった．差分 01 では差分周辺を中心にレビューした被験者のレビュー時間中央値が最も小さくなった．差分 02，03 では差分を中心にレビューした被験者のレビュー時間中央値が最も小さくなった．差分 04 では差分とその周辺をバランスよくレビューした被験者のレビュー時間中央値が最も小さくなった．

図 19 は差分レビューで中心に読んだ場所によって理解度分布を比較したものである．全ての差分において共通に理解度 2 の割合を大きくする読解箇所はなく理解度 2 の割合が大きくなる読解箇所は差分によって違いが表れた．しかしながら全ての差分において差分自体を中心に読んだ被験者の理解度 2 の割合は比較的小さくなり，理解度 2 の割合が最も大きくなることはなかった．

表 10 は重点読解箇所による被験者分類毎のレビュー時間，理解度の分布について統計的検定量を計算した結果である．レビュー時間の分布はクラスカル・ウォリス検定によると全ての差分において統計的有意差が見られなかった．また，理解度分布についてもフィッシャーの正確確率検定によると全ての差分において独立性は認められなかった．

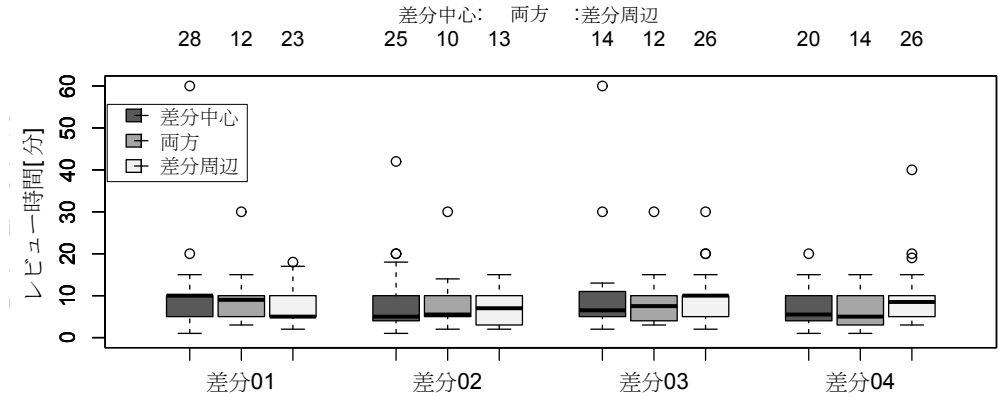


図 18　重点読解箇所によるレビュー時間比較

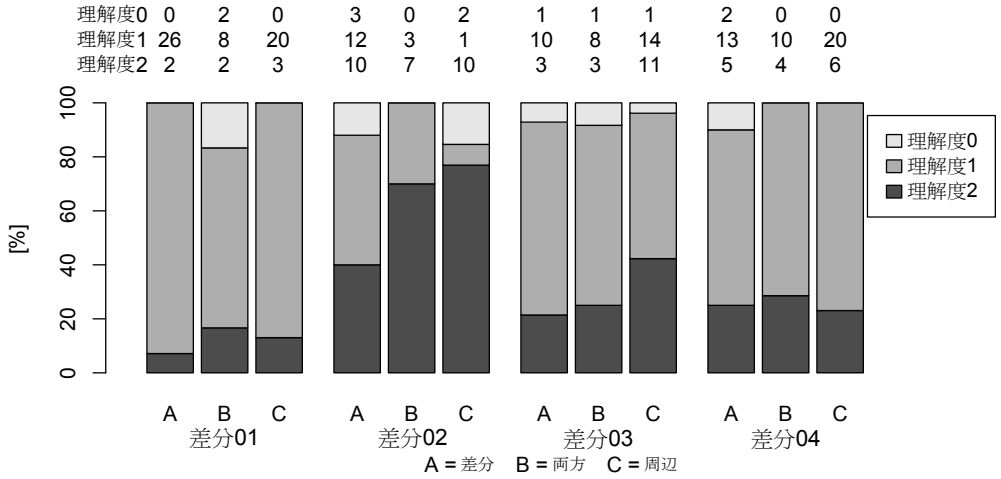


図 19　重点読解箇所による理解度比較

表 10　重点読解箇所によるレビュー時間・理解度の有意差検定結果

検定対象		p 値			
		差分 01	差分 02	差分 03	差分 04
レビュー時間	クラスカル・ウォリス検定	0.835	0.831	0.772	0.249
理解度		0.075	0.065	0.617	0.508

5.4.2 ソースコード表示媒体による比較 (RQ2.2)

図 20 は実験で用いた道具によってレビュー時間分布を比較したものである．全ての差分において PC と紙の両方を用いた被験者のレビュー時間中央値はそれ以外の被験者に比べて比較的大きくなった．また，全ての差分において PC を使ってレビューした被験者のレビュー時間中央値は最も小さくなった．

図 21 は実験で用いた道具によって理解度分布を比較したものである．差分 01, 02 においては紙を用いてレビューした被験者の理解度 2 の割合が PC を用いてレビューした被験者よりも大きくなった．対照的に，差分 03，04 においては PC を用いてレビューした被験者の理解度 2 の割合が紙を用いてレビューした被験者の理解度 2 の割合よりも大きくなった．

表 11 はソースコード表示媒体による被験者分類毎のレビュー時間，理解度の分布について統計的検定量を計算した結果である．レビュー時間の分布はクラスカル・ウォリス検定によると差分 03，04 において統計的有意差が見られた．差分 03，04 の各群をマンホイットニーの U 検定で検定した結果，差分 03 では紙と PC，PC と両方の群間に有意水準 5% で統計的有意差が見られた．一方，理解度分布についてはフィッシャーの正確確率検定によると全ての差分において独立性は認められなかった．

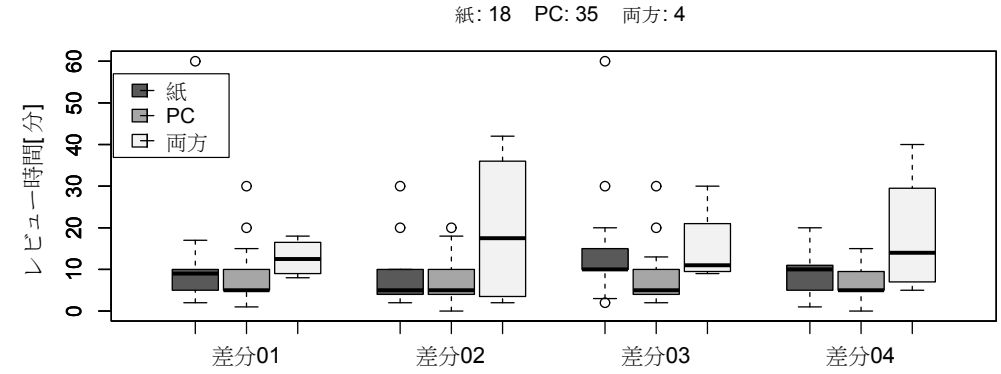


図 20　ソースコード表示媒体によるレビュー時間比較

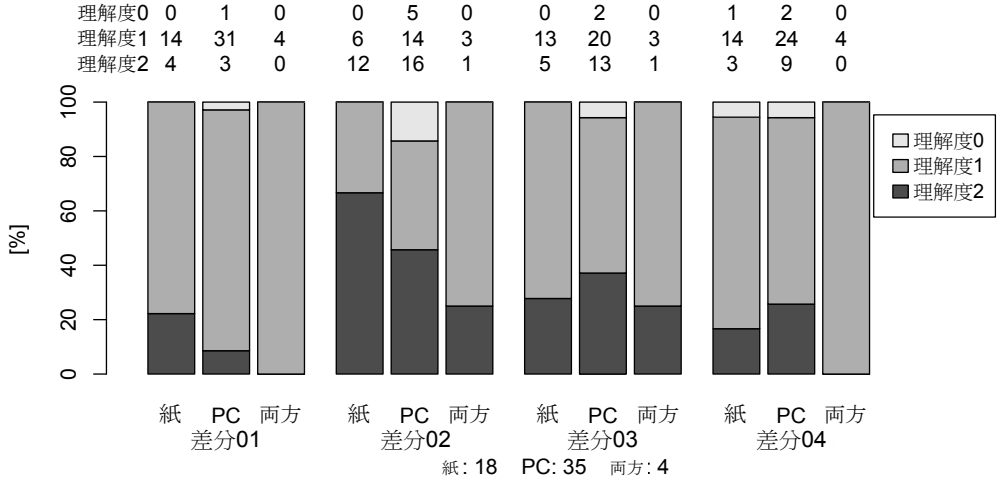


図 21　ソースコード表示媒体による理解度比較

表 11　ソースコード表示媒体によるレビュー時間・理解度の有意差検定結果

検定対象		p 値			
		差分 01	差分 02	差分 03	差分 04
レビュー時間	クラスカル・ウォリス検定	0.146	0.677	0.001 **	0.049 *
	紙 V.S. PC	/	/	0.001 **	0.065
	紙 V.S. 両方	/	/	0.930	0.388
	PC V.S. 両方	/	/	0.023 *	0.056
理解度		0.371	0.200	0.506	0.776

5.4.3 バージョン 1.0 読解方針による比較 (RQ2.3)

図 22 はバージョン 1.0 の読解方針によってレビュー時間分布を比較したものである．差分 01，02 では局所優先理解でバージョン 1.0 を読み進めていた被験者のほうがレビュー時間中央値が小さくなったが，差分 03，04 では全体優先理解でバージョン 1.0 を読み進めていた被験者のほうがレビュー時間中央値が小さくなった．

図 23 はバージョン 1.0 の読解方針によって理解度分布を比較したものである．全ての差分において共通してバージョン 1.0 の読解方針が局所優先理解とした被験者の方が全体優先理解とした被験者の理解度 2 の割合よりも大きくなった．

表 12 はバージョン 1.0 読解方針による被験者分類毎のレビュー時間，理解度の分布について統計的検定量を計算した結果である．マンホイットニーの U 検定によると全ての差分において統計的有意差が見られなかった．また，理解度分布についてもフィッシャーの正確確率検定によると全ての差分において独立性は認められなかった．

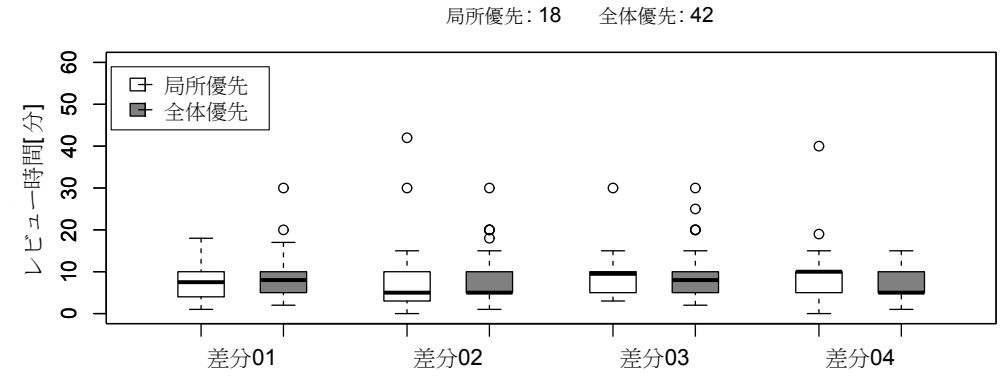


図 22　バージョン 1.0 の読解方針によるレビュー時間比較

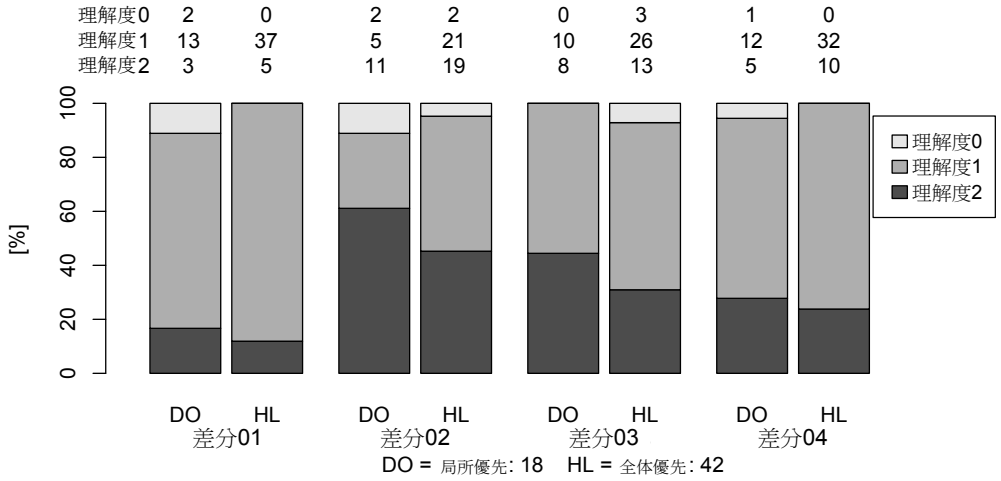


図 23　バージョン 1.0 の読解方針によるレビュー理解度比較

表 12　バージョン 1.0 読解方針によるレビュー時間・理解度の有意差検定結果

検定対象		p 値			
		差分 01	差分 02	差分 03	差分 04
レビュー時間	局所優先理解 V.S. 全体優先理解	0.735	0.786	0.837	0.356
理解度		0.141	0.235	0.504	0.358

5.5 検定結果のまとめ

本節では被験者の開発経験の違い及び読解法の違いによる差分レビュー時間比較及び理解度比較において統計的検定量の計算結果をまとめたものを示す.

表 13 に被験者の開発経験の違い及び読解法の違いによるレビュー時間比較の統計的検定量の計算結果をまとめる．被験者の開発経験の違い及び読解法の違いによるレビュー時間比較の検定にはクラスカル・ウォリス，ウィルコクソンの順位和検定を用いた.

表 14 に被験者の開発経験の違い及び読解法の違いによる理解度比較の統計的検定量の計算結果をまとめる．被験者の開発経験の違い及び読解法の違いによる理解度比較にはフィッシャーの直接確率計算を用いた.

表 13 レビュー時間の有意差検定

カテゴリ		検定対象	p 値			
			差分 01	差分 02	差分 03	差分 04
経験	主言語	クラスカル・ウォリス検定	0.411	0.320	0.539	0.387
	Java/Swing ライブラリ利用経験	経験なし V.S. 経験あり	0.352	0.459	0.047	0.197
	パッチファイル利用経験	クラスカル・ウォリス検定	0.185	0.203	0.014 *	0.221
		経験なし V.S. 知識のみ	/	/	0.019 *	/
		経験なし V.S. 利用経験あり	/	/	0.009 **	/
		知識のみ V.S. 利用経験あり	/	/	0.294	/
	過去に携わったソフトウェア規模	クラスカル・ウォリス検定	0.002 **	0.018 *	0.130	0.042 *
		10KLOC 以下 V.S. 10K-1MLOC	0.002 **	0.006 **	/	0.016 *
		10KLOC 以下 V.S. 1MLOC 以上	0.006 **	0.019 *	/	0.035 *
		10K-1MLOC V.S. 1MLOC 以上	0.056	0.778	/	0.661
	プログラミング経験	5 年未満 V.S. 5 年以上	0.848	0.492	0.529	0.314
読解法	重点読解箇所	クラスカル・ウォリス検定	0.835	0.831	0.772	0.249
	ソースコード表示媒体	クラスカル・ウォリス検定	0.146	0.677	0.001 **	0.049 *
		紙 V.S. PC	/	/	0.001 **	0.065
		紙 V.S. 両方	/	/	0.930	0.388
		PC V.S. 両方	/	/	0.023 *	0.056
	バージョン 1.0 読解法	局所優先理解 V.S. 全体優先理解	0.735	0.786	0.837	0.356

表 14 理解度の有意差検定

カテゴリ		差分 01	差分 02	差分 03	差分 04
経験	主言語	0.673	0.953	0.873	0.690
	Java/Swing ライブラリ利用経験	0.708	0.630	0.342	0.592
	パッチファイル利用経験	0.560	0.916	0.683	0.890
	過去に携わったソフトウェア規模	0.519	0.587	0.774	0.396
	プログラミング経験	1	1	0.649	1
読解法	重点読解箇所	0.075	0.065	0.617	0.508
	ソースコード表示媒体	0.371	0.200	0.506	0.776
	バージョン 1.0 読解法	0.141	0.235	0.504	0.358

6. 考察

6.1 レビューアの開発経験がレビュー時間，理解度に与える影響 (RQ1)

RQ1「レビューアの開発経験はレビュー時間・理解度にどのような影響を与えるか？」への回答は実験結果より，レビューアの開発経験はレビュー時間を減少させるが，理解度の向上には影響を与えないと考えられる．以降で，具体的に考察する．

6.1.1 主言語 (RQ1.1)

5.3.1 節に示した結果より，オブジェクト指向型プログラミング言語を主に利用する被験者のレビュー時間は短くなる傾向が示され，レビュー時間分布の中央値は手続き型プログラミング言語を主に利用する被験者の約半分となった．これより，レビュー対象のソースコードに用いられているプログラミング言語の利用経験はレビュー時間の大幅な短縮の要因となることが考えられる．一方で，差分 01，02 においては手続き型プログラミング言語を主に利用する被験者の方がオブジェクト指向型プログラミング言語を利用する被験者よりも理解度 2 の割合が大きくなった．差分 03，04 ではオブジェクト指向型プログラミング言語を利用する被験者のほうが手続き型プログラミング言語を利用する被験者よりも理解度 2 の割合が大きくなった．レビュー対象のプログラミング言語を主に利用していても全ての差分で理解度を高くするわけではなく，差分によって変化すると考えられる．本実験では差分 01，02 の様な変更箇所数が一箇所しかないもの，差分 03，04 の様な変更箇所数が十箇所以上になるものとで違いが表れた．オブジェクト単位での設計分離の特徴を考えると，オブジェクト指向型プログラミング言語を主言語とする被験者の経験は変更箇所数が分散する場合に有効であると考えられる．

6.1.2 Java/Swing ライブラリ利用経験 (RQ1.2)

5.3.2 節に示した結果より，差分 01，02，03 では，Java/Swing ライブラリ利用経験のある被験者の方がそうでない被験者よりもレビュー時間中央値が小さくなる傾向が示され，特に差分 01，03 では Java/Swing ライブラリ利用経験のある被験者のレビュー時間中央値はそうでない被験者に比べ約半分になることを示した．差分 01，02，03 は Swing API を利用した GUI の変更が行われており，このことからクラスライブラリや API，フレームワークなどの利用経験がそれらに関係したレビュー対象をレビューする際にレビュー時間短縮の要因になると考えられる．しかしながら，理解度に関しては Swing 利用経験があっても必ずしも理解度が大きくなるとは限らなかったことからこれらの経験は理解度の向上には大きな影響を及ぼさないと考えられる．

6.1.3 パッチファイル利用経験 (RQ1.3)

5.3.3 節に示した結果より，パッチ利用経験があるもしくはパッチがどのようなものかを知っている被験者のレビュー時間中央値はパッチ利用経験がない被験者のレビュー時間中央値よりも小さくなる傾向が示され，特に差分 03，04 ではパッチ利用経験も有している被験者のレビュー時間中央値が最も小さくなることが示された．このことから，パッチ利用経験やパッチに関する知識がレビュー時間の短縮要因になることが考えられるが，変更されたコードのセグメント数が小さい場合や，それらが密接に配置されている場合においては経験や知識を持っている被験者とそうでない被験者のレビュー時間の差は小さくなると考えられる．一方で，結果よりパッチ利用経験やパッチに関する知識の有無は理解度の向上には影響が小さいと考えられる．

6.1.4 過去に携わったことのあるソフトウェアの規模 (RQ1.4)

5.3.4 節に示した結果より，すべての差分において 10KLOC 以上のソフトウェア開発に携わった経験のある被験者は 10KLOC 未満のソフトウェア開発にしか携わったことがない被験者に比べてレビュー時間中央値が小さくなる傾向が示さ

れ，特に差分 02，04 では携わったことのあるソフトウェアの規模が大きいほどレビュー時間中央値が小さくなる傾向が示され，10KLOC 未満のソフトウェア開発にしか携わったことのない被験者のレビュー時間中央値に比べ約半分となった．このことから，10KLOC 経験していることがレビュー時間短縮の要因となることが考えられるが，理解度に関しては過去に携わったソフトウェアの規模が大きくてもかならずしも理解度 2 の割合が大きくなるとは限らなかった．

6.1.5 プログラミング経験 (RQ1.5)

5.3.5 節に示した結果より，差分 03，04 において 5 年以上のプログラミング経験を持つ被験者のレビュー時間中央値が 5 年未満しかプログラミング経験を持たない被験者のレビュー時間中央値に比べ約半分となることが示された．一般的なプログラミング経験があるとレビュー時間の短縮が期待できると考えられ，特に変更規模が大きい又は変更箇所数が多い差分のレビューにおいてはその差が顕著になると考えられる．一方で，理解度の比較ではプログラミング経験が 5 年以上でも 5 年未満でも大きく差は現れていないことから，プログラミング経験の理解度向上に対する影響は小さいと考えられる．

6.2 レビューアの読解法がレビュー時間，理解度に与える影響 (RQ2)

RQ1「レビューアの読解法はレビュー時間・理解度にどのような影響を与えるか？」への回答は実験結果より，レビューアの読解法は理解度の向上に寄与するが実験に用いた道具を除いてはレビュー時間の減少には影響が小さいといえる．また結果からは理解度を向上させる読解法は差分によって異なることが示された．以降で具体的に考察する．

6.2.1 重点読解箇所 (RQ2.1)

5.4.1 節に示した結果より，差分 01，04 では読解アプローチの違いによって理解度 2 の割合に大きな差は示されなかったが，差分 02，03 ではパッチ自体を中心

にレビューするよりもパッチやその周辺を中心にレビューした被験者の理解度 2 の割合が大きくなる傾向が示された．差分 01，04 はパッチ適用箇所周辺に指摘されるべき箇所が存在していたため以上の結果が得られたと考えられるが，このことから読解箇所は差分の特徴によって選択されるべきであると考えられる．

6.2.2 ソースコード表示媒体 (RQ2.2)

5.4.2 節に示した結果より，差分 01，02 では紙面上に印刷されたソースコードをレビューした被験者の方が PC スクリーンでレビューした被験者よりも理解度 2 の割合が大きくなった．逆に，差分 03，04 では PC スクリーンでレビューした被験者の方が紙面上に印刷されたソースコードをレビューした被験者よりも理解度 2 の割合が大きくなった．これらから，変更コードセグメントが分散している場合のレビューには PC 上で，そうでない場合は紙面上でといったように変更箇所数の特徴によって使い分けることが理解度の向上につながると考えられる．

6.2.3 バージョン 1.0 の読解方針 (RQ2.3)

5.4.3 節に示した結果より，すべての差分においてバージョン 1.0 の読解方針が局所優先理解だった被験者の理解度 2 の割合が大きい結果が示されたが，差分 02，03 ではその差が大きく，差分 01，04 では比較的小さくなった．結果からは理解度を向上させるにはバージョン 1.0 を局所優先理解で読解する方が良いと考えられるが，差分 04 などを考えると局所優先理解と全体優先理解で理解度の差が大きくないこと，全体優先理解のほうがレビュー時間は短いこと等を考えれば，求められる品質やコストの制限などのシチュエーションによってバージョン 1.0 の読解方針を決定することが効率的であると考えられる．

6.3 制約

被験者の経験に関するアンケートの選択項目は他選択項目から独立しているため，項目選択時は各被験者の主観的なバイアスの影響を受けないと考えられるが，

選択項目内での被験者の経験の多少には言及できていない．例えばSwing利用経験があると答えた被験者の中には豊富な経験を持っている者とそうでない者がいると考えられる．その為，被験者の経験分類方法はさらに改善する必要がある．

結果の統計的検定料計算結果からはレビュー時間，理解度比較共に有意な差が認められないものが多数となった．被験者の分類によってサンプル数を多く確保出来なかったことも理由の一つであると考えられるが，もう一つの大きな理由としてレビュー時間の計測方法が考えられる．実験中の被験者によるレビュー時間の計測は必ずしも正確ではなく，大まかな計測時間で報告する被験者がほとんどであった．そのため，正確な時間分布を得ることができず，帰無仮説の棄却には至らなかったと考えられる．しかしながら，本論文で得られた結果は多くの実務者から得られた傾向であり，またそこから得られた大まかな傾向は実際のソフトウェア開発現場においても一般的な傾向であることが期待できる．

差分をレビューする順番は指定していないため，被験者各々に様々な学習バイアスがかかっている可能性が考えられる．しかし，各差分の変更箇所の重複は少ないためレビューに対する影響は十分に小さいと考えられる．

本論文でレビュー対象としたアプリケーションのソースコード行数はワークショップの限られた時間内で実験を施行するための設計であるが，社会一般のアプリケーションと比較するとその規模は小さい．そのため，本論文で得られた結論はより大きな規模のソフトウェア開発における工数見積もりの面では限界があると考えられる．しかしながら，対象のアプリケーションはデザインパターン，幾何学的アルゴリズム，イベント駆動型アーキテクチャ，オブジェクト指向プログラミングといった本質的要素を含んでおり，被験者の経験や読解アプローチによるレビュー時間，理解度を検証するのには十分であったと考えられる．また，小規模のアプリケーションを用いる利点はより多くの実務者の参加を促すことであり，本論文のように多くの実務者を対象とした実験結果から得られた傾向は一般化を可能にするものであると期待している．

7. まとめ

レビューアの経験や読解アプローチがソフトウェア保守開発におけるレビュー時間，理解度に与える影響を調査するため，ソフトウェア開発の実務者を被験者とした検証実験を実施した．被験者には既存ソースコードとしてGUIアプリケーションのソースコード（バージョン1.0）を読解してもらい，バージョン1.0読解方針を記述してもらった．その後バージョン1.0に変更を加えたパッチ形式のソースコード差分4つを自然言語やスクリーンキャプチャ図から構成される変更タスクドキュメントを用いながらレビューしてもらった．差分レビュー後にはレポートとして差分が変更仕様を満たしているかの差分適応可否とその判断理由，また差分読解箇所も記録してもらった．

本論文では読解方針の記述から二つの読解方針に分類し，各差分の適応可否とその判断理由から理解度を三段階に分類し，被験者の経験（五種類）とアプローチ（実験に用いた道具，差分の中心読解箇所）をアンケートから収集した．また，実験で得られたレビュー時間，分類した理解度を五つの経験や三つの読解アプローチによって比較した．

4つの差分全てのレビューを完了した被験者65人分の実験結果から，レビューアの経験はレビュー時間短縮には大きな影響を及ぼすがレビューアの理解度向上には影響が小さいということが示された．結果ではレビュー対象に用いられているプログラミング言語（Java：オブジェクト指向型プログラミング言語）の利用経験がレビュー時間短縮に最も影響があることが分かった．また，結果からレビューアの読解アプローチはレビューアの理解度の向上には大きく影響を及ぼすがレビュー時間短縮には影響が小さいということも示された．結果では適切な読解アプローチは差分の特徴に依存することが示され，変更されるコードセグメントが一箇所の場合，PCを用いてレビューするよりも紙に印刷されたソースコードをレビューする方が理解が深くなることや，逆に変更されるコードセグメントが分散している場合は上に印刷されたものよりもPCスクリーン上でレビューする方が理解が深くなる傾向などが得られた．

結果からはレビュー対象のプログラミング言語を主言語としているかがレビュー時間の短縮に最も影響を及ぼすが，理解度向上には差分によって適切なアプロー

チで読解することが影響しているという知見が得られた．このことから，レビュー時間や品質の見積りのためにはレビューアの経験や読解アプローチを考慮する必要があると考えられ，効率化のためにはこれら二つの要素を同時に考えなければならないことが分かる．まず，可能な限りレビュー時間を削減したい場合には主に用いているプログラミング言語の種類やプログラム中に用いられるライブラリの利用経験を考慮しなければならないこと，次に理解度向上のためには差分の特徴によって適切な読解アプローチを選択しなければいけないことが結果より提案できる．また今回の結果は経験の乏しいレビューアが身につけるべき経験の優先度を示唆しており例えばプログラミング言語やフレームワーク及びクラスライブラリの利用経験が学習の優先事項になることを示している．

課題としては開発経験の測定方法が挙げられる．今回の測定方法では各分類内で被験者間の経験差を考慮することができなかった．そのため，各被験者間の経験差を埋めることができる開発経験の計測法の考案が大きな課題である．これによってレビューアの開発経験を正確に把握することができれば，より詳細な分析が可能となり，どのような経験が最も重要であるかをタスクごとに評価することが可能であると考えられる．

謝辞

本研究を遂行するにあたり多くの方々に御指導，御助言，御協力を賜りました．この場を借りてこれまでお世話になった方々に感謝の意を表します．ありがとうございました．

主指導教員であり本論文の審査委員を務めて頂いた，奈良先端科学技術大学院大学情報科学研究科松本健一教授に対し，厚く御礼申し上げます．論文執筆やプレゼンテーションの作法など研究に対する直接的な指導に限らず，研究に取り組む上での心構えから，研究者としての姿勢などありとあらゆる面で熱心な御指導を賜りました．心より感謝致します．

副指導教員であり本論文の審査委員を務めて頂いた，奈良先端科学技術大学院大学情報科学研究科飯田元教授には，本研究を進めるに当たり，貴重な御指導を賜りました．学内発表において多数の御質問，御指摘を頂き，心より感謝致します．

副指導教員であり本論文の審査委員を務めて頂いた，奈良先端科学技術大学院大学情報科学研究科門田暁人准教授には，本論文に対して多くの建設的なアドバイスを賜りました．また，研究以外の様々な活動においてもお世話になりました．心より感謝致します．

副指導教員であり本論文の審査委員を務めて頂いた，静岡大学情報学部情報社会学科森崎修司助教には，私の研究生活の全過程において熱心な御支援を賜りました．研究の進め方から，論文執筆，プレゼンテーション技術，全てに渡り，熱意ある御指導を賜りました．私の研究は森崎先生の御指導がなければ成し得ませんでした．また，社会人としての心構えも自身の経験を踏まえて熱心にご指導して頂きました．これからの私の人生においても大変得難い経験となりました．心より感謝致します．

奈良先端科学技術大学院大学情報科学研究科大平雅雄助教には，有益な御指摘，御意見を頂きました．また，物事を進める上で段取りの重要さや，締め切りを守ることへの心構えなど，数多くの御指導を賜りました．研究以外の様々な活動においても，しばしばお気遣いの言葉を賜り，私の学生生活をサポートしていただきました．心より感謝致します．

奈良先端科学技術大学院大学情報科学研究科 Mike Barker 教授には，英語論文

の執筆にあたり英語の添削や論文構成に関するアドバイスを賜りました．心より感謝いたします．

奈良先端科学技術大学院大学情報科学研究科ソフトウェア工学講座伊原彰紀氏にはデータ分析の観点や，統計的検量の計算方法，使用法など研究を進めるに当たって具体的なアドバイスを賜りました．心より感謝いたいします．

奈良先端科学技術大学院大学情報科学研究科ソフトウェア工学講座角田雅照助教には研究のみならず，食生活の面で多大な支援を賜りました．研究面では研究用 PC や周辺機器の充実にご支援を賜りました．食生活面では特に，餃子大食い大会においては筆頭スポンサーになっていただき，運営委員長及び参加者の一人として心より感謝致します．

奈良先端科学技術大学院大学情報科学研究科ソフトウェア工学講座乾氏には学会への出張申請の手続きなど事務的な手続きをサポートして頂き，自身の研究に集中することができました．心より感謝いたいします．

奈良先端科学技術大学院大学ソフトウェア工学講座ならびにソフトウェア設計学講座の皆様には，日頃より多大な御協力と御助言を頂きました．研究発表の準備や研究の過程で賜った御助力，励ましの言葉により，私は研究を完遂することができました．研究のみならず日々の生活においても，皆様の明るさ，優しさにより非常に充実した二年間を過ごすことができました．深く感謝致します．

本論文で実施したソースコードレビューワークショップは IBM アカデミック・イニシアティブ（将来の IT 技術者の育成を目的とした，学生の IT スキル向上支援プログラム：http://www.ibm.com/developerworks/university/academicinitiative/）の支援を受けて開催されました．

参考文献

[1] Marwen Abbes, Foutse Khomh, Gueheneuc Yann G., and Giuliano Antoniol. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In *Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, pages 181–190, 2011.

[2] Evelyn J. Barry, Chris F. Kemerer, and Sandra A. Slaughter. On the uniformity of software evolution patterns. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*, pages 106–113, 2003.

[3] Hans C. Benestad, Bente Anda, and Erik Arisholm. Understanding cost drivers of software evolution: a quantitative and qualitative investigation of change effort in two evolving software systems. *Empirical Software Engineering*, 15(2):166–203, April 2010.

[4] Jean M. Burkhar, Francoise Détienne, and Susan Wiedenbeck. The effect of object-oriented programming expertise in several dimensions of comprehension strategies. In *Proceedings of the 6th International Workshop on Program Comprehension*, pages 82–89, 1998.

[5] G Canfora and A Cimitile. *Handbook of Software Engineering and Knowledge Engineering*, chapter Software maintenance, pages 91–120. World Scientific, River Edge NJ, 2001.

[6] Thomas A. Corbi. Program understanding: challenge for the 1990's. *IBM Systems Journal*, 28(2):294–306, 1989.

[7] Bas Cornelissen, Andy Zaidman, Arie V. Deursen, Leon Moonen, and Rainer Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Transactions on Software Engineering*, 99(5):684–702, 2009.

[8] Cynthia L. Corritore and Susan Wiedenbeck. Mental representations of expert procedural and object-oriented programmers in a software maintenance

task. *International Journal of Human Computer Studies*, 50(1):61–83, 1999.

[9] Cynthia L. Corritore and Susan Wiedenbeck. An exploratory study of program comprehension strategies of procedural and object-oriented programmers. *International Journal of Human Computer Studies*, 54(1):1–23, 2001.

[10] Alastair Dunsmore, Marc Roper, and Murray Wood. Object-oriented inspection in the face of delocalisation. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 467–476, 2000.

[11] Alastair Dunsmore, Marc Roper, and Murray Wood. The role of comprehension in software inspection. *The Journal of Systems and Software*, 52(2-3):121–129, 2000.

[12] Alastair Dunsmore, Marc Roper, and Murray Wood. Systematic object-oriented inspection - an empirical study. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 135–144, 2001.

[13] Alastair Dunsmore, Marc Roper, and Murray Wood. The development and evaluation of three diverse techniques for object-oriented code inspection. *IEEE Transactions on Software Engineering*, 29(8):677–686, 2003.

[14] Michael E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976.

[15] IEEE. *IEEE Standard for Software Maintenance, IEEE Std 1219-1998*, volume 2. IEEE Press, 1999.

[16] Amela Karahasanović, Annette K. Levine, and Richard C. Thomas. Comprehension strategies and difficulties in maintaining object-oriented systems: An explorative study. *The Journal of Systems and Software*, 80(9):1541–1559, 2007.

[17] Amela Karahasanović and Richard C. Thomas. Difficulties experienced by students in maintaining object-oriented systems: an empirical study. In

Proceedings of the 9 th Australasian Conference on Computing Education, pages 81–87, 2007.

[18] Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(12):971–987, 2006.

[19] Corritore Cynthia L. and Wiedenbeck Susan. Direction and scope of comprehension-related activities by procedural and object-oriented programmers: An empirical study. In *Proceedings of the 8th International Workshop on Program Comprehension*, pages 139–148, 2000.

[20] Mika V. Mantyla and Casper Lassenius. What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering*, 35(3):430–448, 2009.

[21] David L. Parnas. Software aging. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 279–287, 1994.

[22] Adam A. Porter and Lawrence G. Votta. An experiment to assess different defect detection methods for software requirements inspections. In *Proceedings of the 16th International Conference on Software Engineering*, pages 103–112, 1994.

[23] Darrell R. Raymond. Reading source code. In *Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '91, pages 3–16. IBM Press, 1991.

[24] Martin P. Robillard, Wesley Coelho, and Gail C. Murphy. How effective developers investigate source code: An exploratory study. *IEEE Transactions on Software Engineering*, 30(12):889–903, 2004.

[25] Harvey Siy and Lawrence G. Votta. Does the modern code inspection have value? In *Proceedings of the 9th IEEE International Conference on Software Maintenance*, pages 281–289, 2001.

[26] Elliot Soloway and Kate Ehrlich. Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, 10(5):235–267, 1984.

[27] Margaret A. Storey. Theories, methods and tools in program comprehension: Past, present and future. In *Proceedings of the 13th International Workshop on Program Comprehension*, pages 181–191, 2005.

[28] Susan Wiedenbeck and Vennila Ramalingam. Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human Computer Studies*, 51(1):71–87, 1999.