Vol.2014-SE-185 No.1
2014/7/9

コードレビュー時の読み方教示によるレビュー効率の変化

應治 沙織[1]　上野 秀剛[1]

概要：ソースコードレビューは人の手に依る作業のため自動化が難しく，作業効率を向上させることが重要である．本稿では，レビュー効率の高い作業者の読み方を他の被験者に教示することで，バグ発見効率が向上するか調査する．実験では被験者を 2 つのグループに分割し，片方のグループにはレビュー開始直後にソースコード全体を見渡し，その後設計書とソースコードの整合性を比較するよう指示した．実験の結果，読み方を教示したグループは教示無しのグループと比べて有意に高いレビュー効率を示し，最大で 20.4%の差が見られた．

1. はじめに

ソースコードレビュー（以後，コードレビュー）とはソフトウェアレビューの一種で，プログラムの中に混入しているバグを検出するために，開発者がソースコードを精読することである．通常，コードレビューは作業（レビュアー）がプログラムをコンパイルや実行すること無しに，画面上，あるいは紙に印刷したソースコードに対して行う．ソフトウェア開発において結合テストやシステムテストのような後工程で誤りが見つかった場合，その除去には多くの労力とコストがかかる．そのため，実行可能なコードが存在しない，開発初期に実施可能なコードレビューは誤りの早期発見と開発コストの削減に有用である．

ソフトウェアレビューの効率を向上させるために，これまでに多くの研究が行われ，様々な手法が提案されている．代表的な手法としては，過去のレビュー経験に基づいて作成したチェックリストを用いる Checklist-Based Reading(CBR)[1] や，複数の異なる立場や視点からレビューする Perspective-Based Reading(PBR)[2]，ユーザの視点からレビューする Usage-Based Reading(UBR)[3] などがある．これらの手法はいずれもレビュー対象のドキュメントに対して，どのような基準で読み進めるか，またはどのような特徴に着目して読むかといったレビュー基準を定めることで，レビューの効率・効果を高める点に特徴がある．一方で，従来研究の結果を比べると，同じ手法を用いたレビューにおいても，その効果・効率には研究ごとに大幅な差が見られる．これは，従来研究で提案されているレビュー手法が大まかな方針のみを指示しているため，被験者によってその解釈や理解度が異なってしまい，レビュー手法の意図通りにレビューした被験者とそれ以外の被験者でレビュー効率に差が出たと考えられる．

そこで，本研究では被験者間で解釈の違いが起きないような，具体的な手順を指示してレビューを実施してもらう．著者の一人が行った過去の研究で得られた，レビュー効率の高い被験者の読み方を教示することでレビュー効率が向上するか調査する．実験では読み方を教示する被験者グループと，教示しない被験者グループにそれぞれレビューを実施してもらい，レビュー効率を比較する．その際，教示したグループが，指示した読み方を実施しているかを確認するために視線移動を計測する．

本論文の構成は以下のとおりである．2 章では関連研究について示す．3 章ではソフトウェアレビューにおける読み方教示について説明する．4 章では実験環境や実験手順などを示し，5 章では実験結果と考察を述べる．6 章ではまとめと今後の課題について述べる．

2. 関連研究

2.1 ソフトウェアレビュー

ソフトウェアレビューの効率を向上させる手法はこれまでにも多数提案されている [1], [2], [3]．代表的な手法として過去のレビュー経験に基づいて作成されたチェックリストを用いる Checklist-Based Reading (CBR)[1] や，対象ソフトウェアに関わる複数の異なる視点（ユーザ，プログラマ，設計者など）からレビューを行う Perspective-Based Reading(PBR)[1]，PBR の一種であり，ユーザの視点からレビューを行う Usage-Based Reading(UBR)[3] などがあり，これらの手法の性能を比較した研究も多い．

松川らは UML で記述された設計書を対象としたレビュー

[1] 奈良工業高等専門学校
National Institute of Technology, Nara College

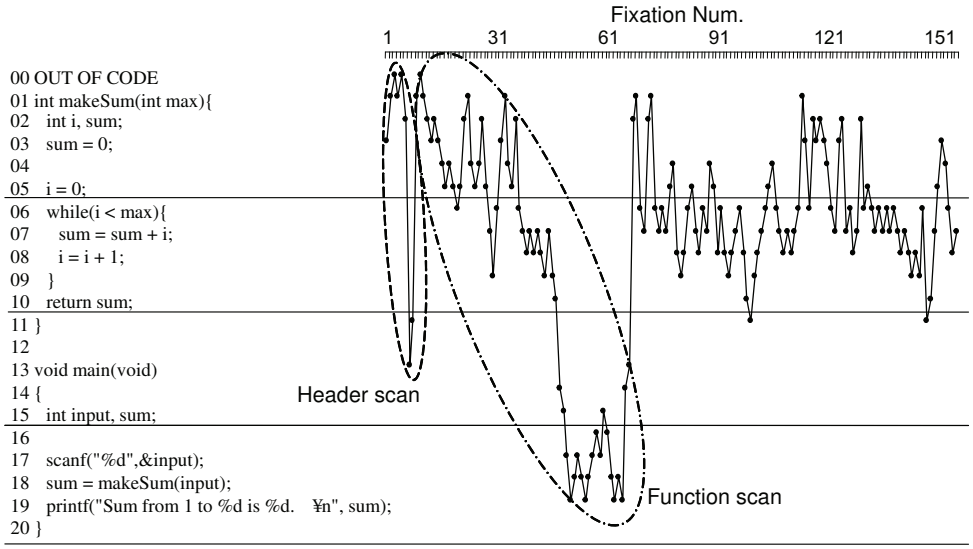

図 1 レビュー作業者の視線移動

実験で，CBR と PBR を比較している [4]．実験の結果は PBR が図と図の関連性や一貫性に関する意味的なバグを発見しやすく，CBR は構文的なバグが発見しやすいことを示している．一方で，手法の性能を比較している研究の結果を比べると研究によって異なる結果を示している場合がある．例えば UBR と CBR を比較した研究では Porter らは UBR のほうが優れている [5] とする一方で，Lanubile らは有意な差はない [6] としており，手法の違いについて明確な結果が得られていない．また，手法による差と比べて，同じ手法を用いた被験者間の差が大きい [7] ことから，レビュー方法の大まかな方針を指示するだけではレビュー効率の向上には必ずしも繋がらない恐れがある．

本研究では，レビュー開始直後における提示物の読み方についてより具体的な指示をすることでのレビュー効率が向上するか調査する．

2.2 視線計測による能力評価

視線移動の計測による被験者の能力比較や行動分析は認知科学の分野においてよく用いられる [8], [9], [10]．村田らは自動車運転時の危険予知における熟練者と初心者の視線の動きを比較している [8]．実験では，自動車運転状態の静止画を運転熟練者と初心者に見せ，危険を予知する際の視線運動を解析している．実験の結果，危険度の高いカーブで熟練者と初心者の視線移動の違いが顕著になる一方で，危険度がカーブよりも相対的に低い直線道路や交通量の少ない道路では熟練者と初心者の視線移動の違いが少なくなった．Law らは腹腔鏡手術の訓練装置を使用している初心者と熟練者の視線移動を分析し，熟練者は初心者に比べて患部を集中して見ていることを明らかにしている [9]．

ソフトウェア開発の分野においても，視線計測による開発者の分析や支援に関する研究が行われている [11], [12], [13], [14]．中道らは Web ページ閲覧時のユーザビリティ評価に視線情報を利用している [12]．Web ページのユーザビリティの評価指標に実験時に記録した視線データを用いることで，Web ページのユーザビリティについてより多くのコメントが得られることを明らかにした．Stein らはソフトウェアのデバッグ被験者に他の被験者の視線移動を見せることが作業の助けになるということを示している [13]．分析結果から，他の被験者の視線移動を見てからデバッグを開始した被験者は視線移動を見なかった被験者よりも早くバグを発見できることを明らかにした．

著者の一人はこれまでにコードレビュー被験者がレビュー中にソースコードのどこに注目しているか視線を計測し，レビュー能力との関係について調べている [14]．被験者に仕様書，設計書，ソースコード，コードチェックリストを提示した実験の結果，レビュー開始時にコード全体を上から下に向かって眺める動作を行わない被験者は誤り検出までの時間が長くなる傾向が見られた．本稿では，レビュー開始時にプログラム全体を見渡すよう指示することでレビュー効率が向上するか調査する．

3. コードレビューにおける読み方の教示

本研究ではコードレビューにおけるレビュー対象物の読み方を被験者に指示することでレビュー効率が向上するか調査する．図 1 に著者らの過去の研究で計測した，レビュー効率が高い作業者の視線の動きをを示す．図の左側はレビュー対象のソースコードを，右側がレビュー開始からの視線移動をグラフで示している．視線移動を示すグラフから，この作業者が開始直後にソースコード全体を見渡している事が分かる．過去の研究から，レビュー効率が高い作業者の視線移動から同様の傾向が見られた．

バグを検出するためにはプログラムの制御フローやソー

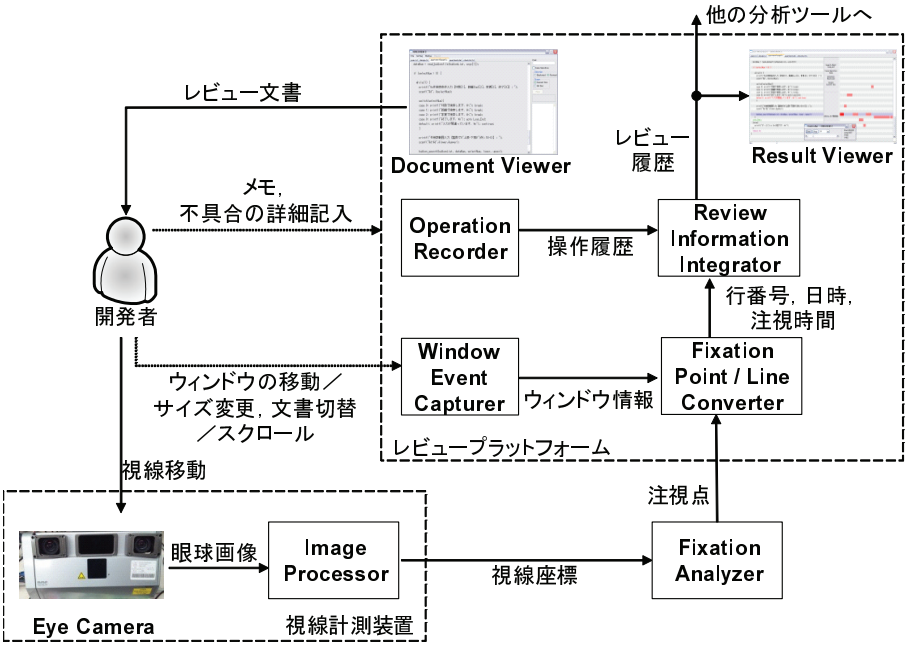

図 2 実験環境

スコード上のクラス・メソッドの記述位置といった構造を理解する必要がある．著者らの過去の研究で見られた視線移動はプログラムの構造をレビュー開始時に確認する動作であると考えられる．レビュー開始時にプログラムの構造やクラス・メソッドの記述位置を把握することは，それ以降のレビューを効率的に実施するために有用である．そこで，本研究では作業者にコードレビュー開始時の読み方を教示し，設計書の内容やソースコードの構造を把握させることで，レビュー効率が向上するという仮説を立てる．

また，設計書を含んだコードレビューの場合，レビュー対象であるソースコードだけでなく，設計書に記述されたプログラムの機能や構造の理解も必要である．設計書を読むことでプログラム自体の理解が効率的になることに加え，設計書とソースコードのずれを見つけることで，効率的にバグを発見できると考えられる．本研究では，レビュー開始時にプログラムの構造を把握することに加え，設計書とソースコードの整合性を確認するような読み方を教示することで，レビュー効率が向上するか確認する．

本研究では，プログラムの設計を理解するための動作，ソースコードの構造を理解するための動作，設計とソースコードの整合性を確認するための動作として以下の 3 手順をレビュー作業者に教示する．

(1) 設計書をよく読んでください
(2) コード全体にざっと目を通してください
(3) 設計書に書かれているフィールド，メソッドが正しくソースコードに実装されているか確認してください

手順 (1) は設計書を読むことでプログラムの設計を理解してもらうための指示である．手順 (2) と (3) はソースコードの構造を理解してもらうとともに，設計書との整合性を確認することでプログラム全体を理解した上でレビュー作業をしてもらうための指示である．本章で述べてきたとおり，これらの手順を教示することで作業者のレビュー効率が向上すると考えられる．一方で，教示を実施することでレビュー作業開始から数分間はバグを検出しなくなり，教示がない場合と比べてレビュー効率が低下する可能性がある．本稿では，教示した被験者と教示しなかった被験者のレビュー効率を比較することでレビュー時の読み方を教示する事の効果を調査する．

4. 実験

実験は予備実験と本実験の 2 回を実施する．本稿ではレビュー効率の指標として，バグ発見率を用いる．予備実験では被験者のバグ発見率を測るため，全ての被験者に読み方を教示せず Java で記述されたソースコードをレビューしてもらう．次に，予備実験で測定した各被験者の誤り発見効率をもとに，バグ発見率の平均がほぼ等しくなるように被験者を 2 つのグループ” 教示有り” と” 教示無し” に振り分ける．本実験では” 教示有り” グループにのみ 3 章で説明した，レビュー開始時の読み方を教示し，” 教示無し” グループとバグ発見率を比較する．実験の被験者として，Java によるプログラミングの経験が 1 年以上ある学生 14 人を対象とする．以降の節で，実験で用いる計測環境および，予備実験・本実験の詳細を説明する．

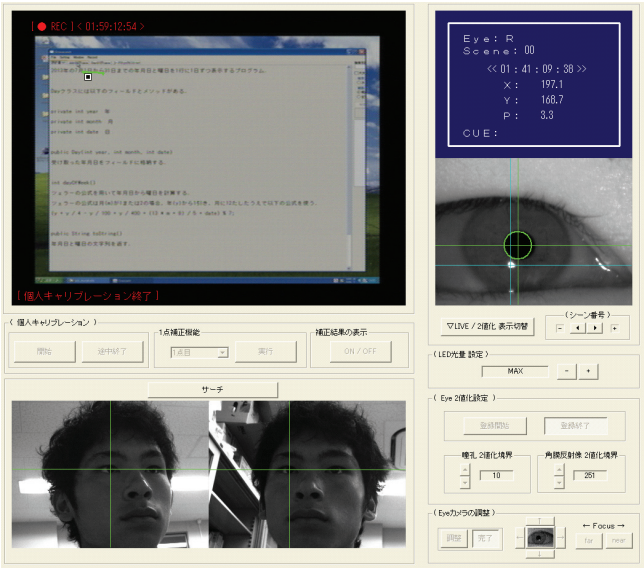

図 3 EMR-AT Voxer Basic+

4.1 実験環境

本研究では，図 2 に示す視線計測環境を用いてレビュー中の視線移動を計測する [14]．本環境は，視線計測装置を含むハードウェアコンポーネントとコードレビュー実験用ソフトウェアから構成される．

4.1.1 ハードウェアコンポーネント

本計測環境は視線計測装置 1 台と，視線計測装置を制御し視線情報を計測する PC1 台，および被験者が利用する PC1 台からなる．実験では上記の構成を 2 組用意し，被験者 2 名を同時に計測した．

視線計測装置は内蔵カメラで撮影した被験者の眼球映像から，ディスプレイ上の注視している座標点を計測する．本研究ではソースコードを精読する被験者の視線移動を計測するため，被験者の視線の動きを行単位で精密に識別する必要がある．そこで本環境では高解像度で精度の高い計測が可能である nac Image Technology 社の非接触アイマークレコーダ EMR-AT Voxer Basic+を用いる．本計測装置は被験者に装置や計測用のめがねなどを装着する必要が無いため，被験者に負担をかけることなく計測が可能である．EMR-AT Voxer Basic+は検出分解能 0.3°，検出レート 60Hz で計測する．計測誤差は当環境においてディスプレイ上で約 5.4pixel となる．これはソースコードを表示したときに，約 0.45 行に相当する．本計測装置で視線を計測している様子を図 3 に示す．

ソースコードを表示するために液晶ディスプレイ（EIZO FlexscanS1721) を解像度 1024 × 768 で使用した．ソースコードは被験者用 PC 上で動作する，レビュープラットフォームを用いてディスプレイに表示される．また，ヘッドレストがついた椅子を用いることでレビュー中に被験者の頭が動かないよう配慮した．

4.1.2 レビュープラットフォーム

視線計測装置が出力するディスプレイ上の座標情報から，レビュー被験者に表示されるレビュー対象文書の行を特定するためにレビュープラットフォーム Crescent (Code Review Evaluation System by Capturing Eye movemeNT)[14] を用いた．Crescent は被験者にレビュー対象となるソースコードと，レビュー時に参照される他の文書（設計書やチェックリスト）をディスプレイに表示すると共に，被験者のウィンドウ移動や表示する文書の切り替え，スクロールなどの操作を計測する．レビュー中に誤りを発見した行をダブルクリックすると，誤り報告用のウィンドウが表示され，誤りの内容と位置を記録できる．

本プラットフォームは操作履歴からディスプレイに表示されている文書の行とその座標を算出し，視線計測装置が出力する座標情報から被験者がどの文書のどの行を見ているか計算する．プラットフォームは被験者の視線がその位置にある行を認識しているか，単にとどまっただけかを区別するために停留点を計算する．停留点とはある時間以上継続して視線が留まった点のことを指し，一般に被験者はその位置にある文字やオブジェクトを意識的に見ていると解釈される．本研究では直径 30pixel の円内に 50ms 以上の視線が留まった場合に，その中心を停留点とみなす．これにより，計測対象者が認識なく対象上を通過したのか，対象を意識して注目したのかを区別することができる．

図 4 に Crescent を用いて計測したレビュー時の視線移動を示す．図の左側はレビュー中に被験者が閲覧した文書を示し，右側には文書の各行に対する視線の停留と移動をバーチャートで示している．また，Crescent は計測した視線移動と被験者の操作を再生し，レビューの様子を再現することができる．

4.2 予備実験

予備実験は被験者のバグ発見率を計測するために実施する．予備実験におけるバグ発見率を元に，バグ発見率が等しくなるよう被験者を 2 つのグループに分類することで，読み方の教示によるバグ発見率の違いを明らかにする．予備実験では，練習を含めた 3 つのプログラムを対象に被験者にレビューを実施してもらう．

表 1 に予備実験で被験者に提示するプログラムを示す．いずれのプログラムも Java で記述されたソースコードと，日本語で記述された設計書からなる．ソースコードにのみ，著者らが複数の誤りを混入した．バグはソースコード 10 から 20 行につき 1 件を混入し，ソースコードの長さからバグの数を予測できないよう配慮した．誤りは，ボーリス・バイザーの作成したバグ分類 [15] に基づいて，発生頻度が高く，特定のプログラム言語に依存しない以下の 3 種類を埋めた．

- 機能不良： 要求と実際のソースコードの相違に起因す

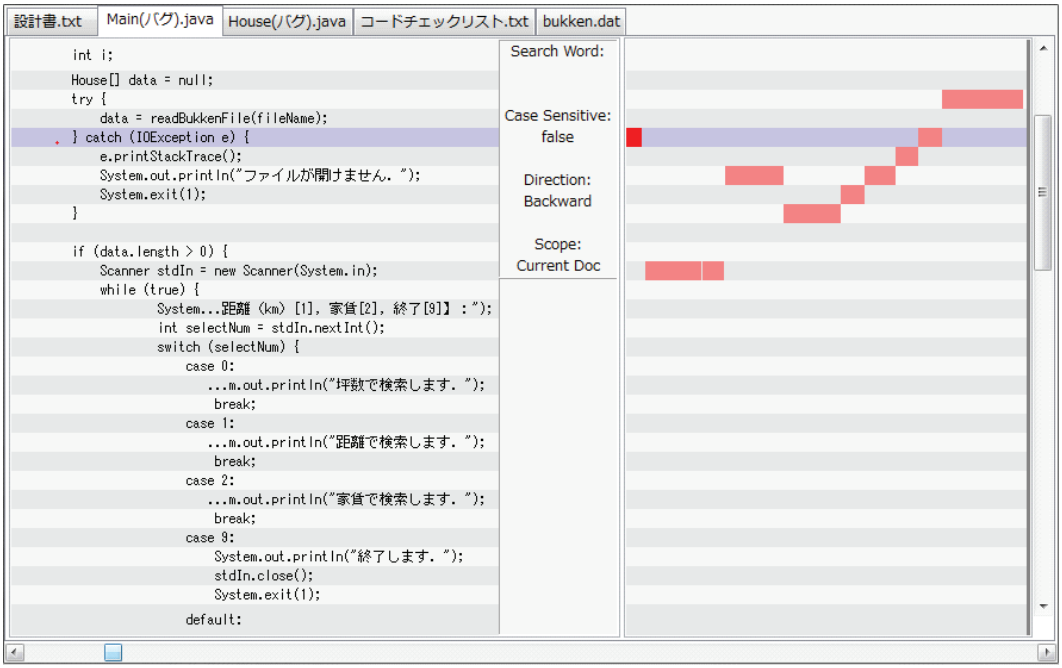

図 4 レビュープラットフォームの出力例

表 1 予備実験で提示するプログラム

	仕様	提示物	行数	バグ数
練習タスク	整数 n を入力すると 1 から n の和を表示する.	Main.java	18	0
		Sum.java	14	1
		設計書	12	–
タスク 1	7 月 1 日から 31 日までの年月日と曜日を 1 行に 1 日ずつ表示する.	Day.java	31	3
		July.java	16	2
		設計書	25	–
タスク 2	複数の銀行口座の残高を表示する.	Account.java	37	5
		Bank.java	41	1
		設計書	37	–

る誤り

- 構造不良： 制御フローやアルゴリズムに起因する誤り
- データ不良： データの仕様や形式，種類，初期値に起因する誤り

全てのレビューにおいて，著者らが作成したコードレビュー用チェックリストを被験者に提示する．チェックリストには，ソースコードの完全性，初期化，メソッド呼び出し，演算，データ・ファイル，制御について正しくコーディングされているかを確認するような項目を用意した．

各被験者は 4.1.2 節で示したレビュープラットフォームを用いてそれぞれのプログラムをレビューする．また，予備実験では設計書，ソースコード，コードチェックリストを読み，ソースコードに含まれる誤りを検出するよう指示した．レビュー時間には制限を設定せず，被験者が誤りをすべて発見した，またはこれ以上の誤りを発見できないと判断したときに終了するよう指示する．バグを検出した際は，誤りが含まれている箇所をレビュープラットホームの機能を用いて被験者に記録させる．検出した誤りの数は，実験終了後にレビュープラットホームが出力する操作履歴をともに著者が手動でカウントする．すべての被験者は最初に練習タスクを実施した後に，2 つのプログラムをレビューする．2 つのプログラムをレビューする順序については，学習効果を考慮して半数の被験者はタスク 1 を最初に実施し，残りの半数はタスク 2 を最初に実施する．

4.3 本実験

本実験はバグ発見率の平均が等しくなるように被験者を 2 つのグループ（教示有り，教示無し）に分割し，教示有りグループにのみ 3 章で示した，レビュー開始時の読み方を教示し，各グループのバグ発見率を比較する．本実験では，練習タスクを含めた 2 つのプログラムを対象に被験者にレビューを実施してもらう．表 2 に本実験で被験者に提示するプログラムを示す．レビューに用いる環境や混入するバグの種類や数，検出したバグの記録方法については予備実験と同様とする．

教示有りグループにのみ，練習タスクと本実験タスクそ

表 2　本実験で提示するプログラム

	仕様	提示物	行数	バグ数
練習タスク	n を入力すると n 段のピラミッドをアスタリスクで表示する.	Pyramid.java	14	1
		Main.java	11	0
		設計書	30	–
タスク	ファイルに記録された賃貸物件から条件に合う物件を検索し表示する.	Main.java	104	6
		House.java	34	1
		bukken.dat	20	0
		設計書	30	–

れぞれの開始前に 3 章に記した教示内容を口頭で被験者に説明した．さらに，被験者が教示した読み方を実施しているか確認するために被験者の視線を計測した．

本実験では教示によってバグ発見率が変化すると考えられるが，レビュー時間を長く取りすぎるとバグ発見率の高い被験者がレビューを終了したにもかかわらずなにもしない時間が生じ，レビューが終了した時点で効率の低い作業者と同様の結果になる恐れがある．バグ発見率の高い（教示をした）被験者がレビューを完了しないと思われる時間にレビュー時間を制限することで，教示によるバグ発見率の違いを正確に分析できる．そこで，本実験ではレビュー時間を 40 分に制限し，時間が経過した時点で作業を終了させた．

5. 結果と考察

5.1 バグ発見率

予備実験における各被験者の率をもとに，被験者を本実験のグループ” 教示有り” と” 教示無し” に分割し，本実験を行った．その結果，教示無しグループのバグ発見率の平均が 62.5%だったのに対し，教示有りグループは 71.4%と 8.9%の差が見られた．これは，レビュー開始時の読み方を教示することで，教示無しの場合と比べてプログラムの設計や構造に対する理解が高くなったためと考えられる．2 つのグループに対して Welch の t 検定（両側検定）を行った結果，$p = 0.11$ で有意な差は見られなかった．

図 5 に教示有りグループと教示無しグループのレビュー時間とバグ発見率の遷移を示す．図の横軸はレビュー開始からの経過時間（分），縦軸はバグ発見率を表している．図から分かる様に，レビュー開始から 10 分までの間，教示有りは教示無しと比べてバグ発見率が低い．また，10 分から 15 分頃までは 2 つのグループのバグ発見率にほとんど差が見られない．しかし，作業開始から 15 分以降は教示有りの方が教示無しに比べてバグ発見率が高く，レビュー開始後 26 分の時点で 20.4%の差が見られた．

教示有りの被験者は教示にしたがって設計書を読んだり，ソースコードの構造を確認していたため，レビュー開始から 10 分間はバグをほとんど検出できていなかったと考えられる．しかし，レビュー開始から 10 分以降はプログラムの設計やソースコードの構造を理解した上でレビューし

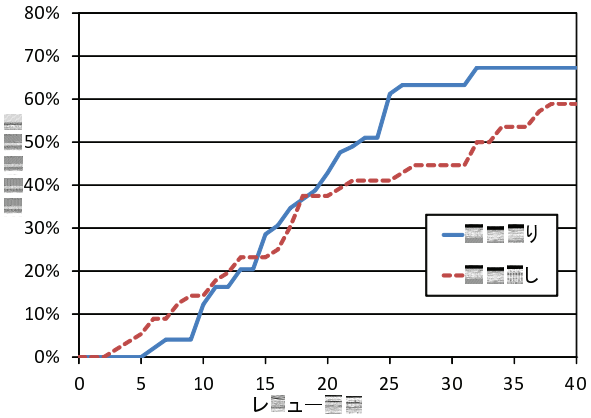

図 5　レビュー時間とバグ発見率

表 3　レビュー対象物に対する被験者の注視割合

	教示有り [%]	教示無し [%]
設計書	28.6	22.8
Main.java	56.7	67.3
House.java	11.8	8.5
bukken.dat	2.3	1.0
コードチェックリスト	0.6	0.4

たため，教示無しのグループに比べて効率的にバグを指摘できたと考えられる．教示有りグループと教示無しグループの分ごとのバグ発見率の差に対して Welch の t 検定（両側検定）を行った結果，25 分から 31 分の範囲において，$p < 0.05$ で有意な差が見られた．

また，レビュー開始から 32 分以降に教示有りグループのバグ発見率が上昇しない一方で，教示無しグループはレビュー終了時間の 40 分までバグ発見率が上昇した．これは，教示有りグループが教示によってプログラムをより素早く理解し，効率的にバグを検出できたため，40 分が経過する前に限界までバグを検出できたと考えられる．一方で，教示無しグループは教示有りグループに比べバグ発見率が低く，レビュー時間中に十分にバグを検出できなかったと考えられる．多くの開発現場においてはコードレビューに長時間かけることは難しいため，より短時間にバグを多く発見できる読み方の教示は有用であるといえる．

5.2 レビュー対象物に対する注視の割合

教示の有無によって設計書やソースコードに対する被験

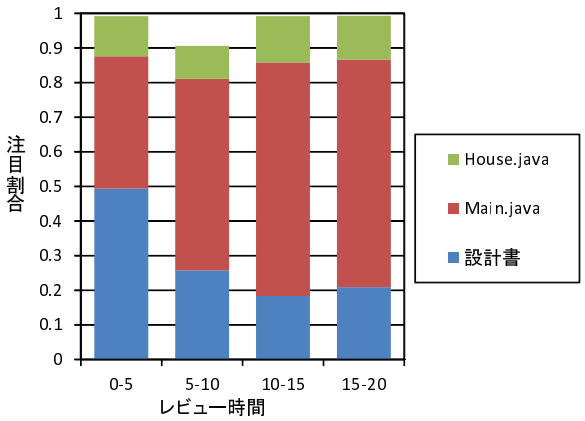

図 6 提示物に対する注視割合（教示有り）

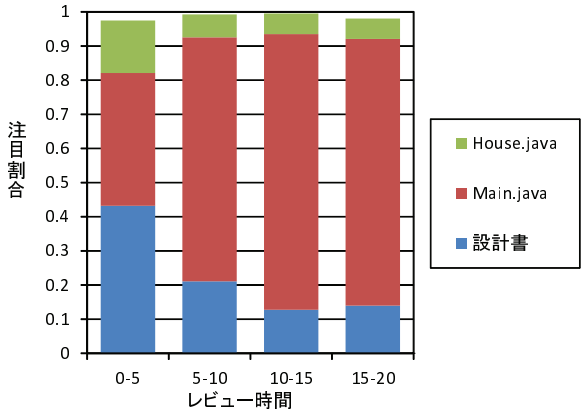

図 7 提示物に対する注視割合（教示無し）

者の注視割合にどれだけ差が出たか確認するために，レビュー開始から 20 分間に被験者が注目したレビュー提示物の割合を分析する．表 3 にレビュー開始から 20 分間における各提示物に対する注視時間の割合を示す．表はいずれのグループにおいても設計書，Main.java，House.java に視線が集中しており，bukken.dat（データファイル）およびチェックリストはほとんど見られていないことを示している．

図 6，図 7 に教示有りグループと無しグループそれぞれの設計書，Main.java，House.java に注目していた割合の変化をそれぞれ示す．いずれのグループも最初の 5 分は設計書を読み，その後 Main.java に多くの時間をかけているが，教示有りグループの方が全期間を通して設計書を 5%から 10%多く注視していることが分かる．これは作業開始直後に設計書をよく読むように指示した事に加え，ソースコードと設計書を比較して見るよう指示したためと考えられる．

また，Main.java に対する注目時間は教示無しのほうが全期間を通じて小さく，House.java に対する注目時間は最初の 5 分間を除いて教示有りのほうが大きい．これは，教示無しのグループは，バグが含まれたソースコードのうち，行数の多い Main.java に集中している一方で，教示有りのグループはそれ以外の提示物（設計書と House.java）に対しても注意を向けていることを示している．House.java にはバグが 1 つ含まれているが，教示有りグループは全員がそのバグを発見した一方で，教示無しグループの 1 名が検出できなかった．この結果は，複数の提示物があるレビューにおいて，全体に目を通すよう促す教示が有効である可能性を示している．

実験で計測した被験者の視線移動を図 8 および図 9 に示す．図 8 は教示有り，図 9 は教示無しの被験者がレビュー開始から 20 分間に注視していた提示物の遷移で，縦軸が提示物を，横軸がレビュー開始からの時間（分）を示す．図から，教示有りの被験者が作業開始時に指示通り設計書を読み，その後，Main.java，House.java, Bukken.dat と順番に提示物全体に目を通していることが確認できる．その後も，2 つのソースコードと設計書を交互に見ながらレビューを進めている様子が確認できる．

一方で，教示無しの被験者は設計書と Main.java を読むことに費やす時間の割合がその他の提示物に比べ大幅に大きく，作業開始から 20 分の間の注意が偏っている．さらに，設計書と Main.java に対する注視でも 2 つのファイルを頻繁に切り替えており，教示有りグループの被験者に比べ設計に対する理解が不十分だった可能性がある．

6. おわりに

本稿ではソースコードレビューにおいて，作業者に対してレビュー開始時に設計書を読むこと，設計書とソースコードの整合性を確認するよう教示することでレビュー効率が向上するか検証した．実験では先行研究で開発した視線計測環境を用い，読み方を教示した被験者と教示しない被験者の視線移動とバグ発見率の遷移を計測した．実験の結果，レビュー開始 25 分から 31 分の範囲において教示有りのグループが教示無しグループよりバグ発見率が有意に高く，最大で 20.4%の差が見られた．これらのことからレビュー作業者に具体的な読み方を教示することは，レビュー効率を向上させる方法として有用であると考えられる．

本稿でレビュー作業者に教示した 3 つの手順は事前の用意や訓練をすることなく実施できる簡単なものである．そのため，開発現場への導入が比較的容易で，特にレビュー経験の浅い作業者に効果的であると考えられる．一方で，レビュー経験が豊富な作業者にとっては既に実施している作業であるため，教示による効果が得られにくい可能性もある．今後，作業者の経験やレビュー対象物に対する事前知識が異なる作業者に対して，教示によってレビュー効率がどの程度向上するか確認する．

また，本稿で実施した教示の内容は，レビュー対象物の精読を始める前にその関連文書の内容を理解させることを

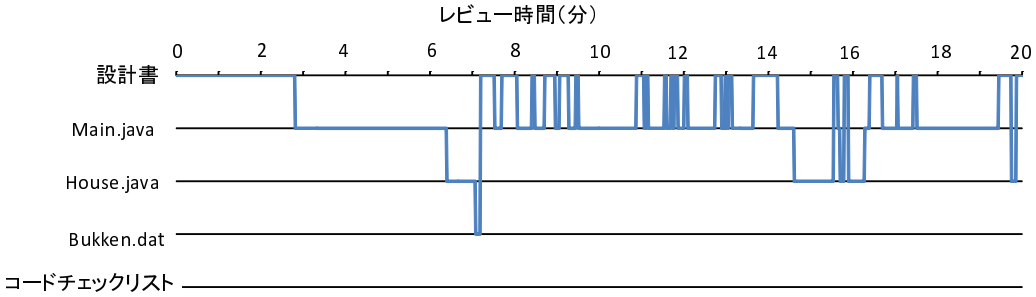

図 8 被験者 c（教示有り）の視線移動

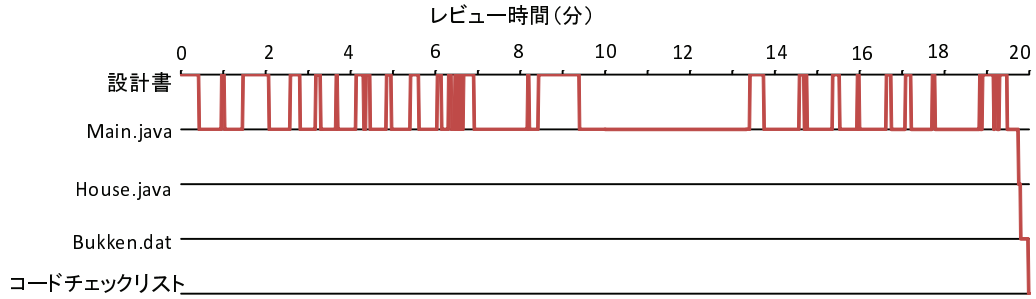

図 9 被験者 j（教示無し）の視線移動

目的としている点で，レビュー前に実施する準備フェーズと類似している点がある．今回の実験では教示内容を実施している間にもバグを検出／指摘することが可能な状態でレビューを実施させているため，厳密には準備フェーズとは異なる作業をしている．今後，同様の作業をレビュー時間外に実施した場合と，レビュー作業の一部として実施した場合のレビュー効率や効果を分析していくことで教示の効果を明確にする必要がある．

謝辞

本研究の一部は JSPS 科研費 若手研究 (B) 24700038 の助成を受けて行われた．

参考文献

[1] M. E. Fagan： “Design and Code Inspection to Reduce Errors in Program Development,” IBM Systems Journal, Vol.15, No.3, pp. 182-211, (1976).
[2] F. J. Shull： “Developing Techniques for Using Software Documents: A Series of Empirical Studies,” PhD thesis, Univ. of Maryland, (1988).
[3] T. Thelin, P. Runeson, and B. Regnell： “Usage-based Reading – An Experiment to Guide Reviewers with Use Cases,” Information and Software Technology, Vol.43, No.15, pp. 925-938, (2001).
[4] 松川文一, Giedre Sabaliauskaite, 楠本真二, 井上克郎：“UML で記述された設計仕様書を対象としたレビュー手法 CBR と PBR の比較評価実験”, オブジェクト指向最前線 2002, pp. 67-74, (2002).
[5] A. A. Porter, L. Votta： “Comparing Detection Methods for Software Requirements Inspection: A Replication using Professional Subjects,” Empirical Software Engineering, Vol.3, No.4, pp. 355-380, (1988).
[6] F. Lanubile, G. Visaggio：“Evaluating Defect Detection Techniques for Software Requirements Inspections,” ISERN pp. 00-08(2000).
[7] T. Thelin, P. Runeson, and C. Wohlin：“An Experimental Comparison of Usage-Based and Checklist- Based Reading,” IEEE Transaction on Software Engineering, Vol. 29, No. 8, pp. 687-704(2003).
[8] 村田厚生, 森若誠：“危険予知課題における運転者の視覚情報処理特性 - 運転初心者と運転熟練者の比較”, 人間工学,Vol.46,No.6, pp. 393-397(2010).
[9] B. Law,M. S. Atkins,A. E. Kirkpatrick,,A. J. Lomax, and C. L. Mackenzie：“Eye gaze patterns differentiate novice and expert in a virtual laparoscopic surgery training environment,” In Proceedings of ACM Symposium of Eye Tracking Research and Applications (ETRA), pp. 41-48,(2004).
[10] 下西慶, 石川恵理奈, 米谷竜, 川嶋宏彰, 松山隆司：“視線運動解析による興味アスペクトの推定”, ヒューマンインタフェース学会論文誌,Vol.16, No.2, pp. 27-38(2014).
[11] S. Zhai,C. Morimoto,and S. Ihde：“Manual and gaze input cascaded (MAGIC) pointing,” In Proceedings of The SIGCHI Conference on Human Factors In Computing Systems ‘99, pp. 246-253(1999).
[12] 中道上, 島和之, 中村匡秀, 松本健一 ：“視線を利用した Web ユーザビリティ評価環境”, 情報処理学会論文誌,Vol.44,No.11,pp. 2575-2586 (2003).
[13] Randy Stein,Susan E. Brennan：“Another Person’s Gaze as a Cue in Solving Programming Problems,”In Proceedings of The 6th International Conference on Multimodal Interface, pp. 9-15(2004).
[14] Hidetake Uwano：“Measuring and Characterizing Eye Movements for Performance Evaluation of Software Review”, Ph.D. thesis, Nara Institute of Science and Technology, (2009).
[15] Boris Beizer：“ソフトウェアテスト技法”(1994).