プログラマの視線を用いたコードレビュー性能の要因分析

Analyzing Performance of Source Code Review Using Reviewers' Eye Movement

上野 秀剛[*] **中村 匡秀**[†] **門田 暁人**[‡] **松本 健一**[§]

あらまし 本論文では視線計測装置を用いて，コードレビュー時のプログラマの視線の動きとレビュー性能（誤り発見効率）の関係を分析するための実験を行った．その結果，プログラマの視線に（1）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，（2）変数が初めて参照されたときに変数宣言部を確認する動作，（3）変数が現れたとき，変数が直前に出現した行を確認する動作の3つのパターンがあることがわかった．さらに，レビュー開始時に（1）の動作を十分に行わなかったとき，誤り検出までの時間が長くなる傾向にあることがわかった．

1 はじめに

ソースコードレビュー（以後，コードレビューとする）とはソフトウェアレビューの種類の1つで，開発中のプログラムのソースコードを精読する作業のことである．その目的はプログラムの開発過程において混入されたソースコード上の誤り（バグ）を発見・除去することである [1]．通常，コードレビューはレビュー作業者がプログラムのコンパイルや実行をせずに，画面上，あるいは紙上のソースコードに対して行われる．コードレビューにおいて作業者はコードを読み，その動作を理解し，誤りを発見した場合，除去を行うかバグ報告書に記録する．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行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ってしまうため開発初期に行うことのできるコードレビューは非常に重要である．

従来，ソフトウェアレビューの効率を高めるために複数のレビュー手法が提案され，それらの性能を比較する実験が行われている．しかし，どの手法が最もレビュー効率（誤り発見確率や誤り発見効率）が良いのかについては明確な答えが出ていない．これまで，Checklist-Based Reading (CBR) は Ad-Hoc Reading (AHR) と同程度の効率しかないことが確認され，Usage-Based Reading (UBR)，Perspective-Based Reading (PBR)，Defect-Based Reading (DBR) については CBR や AHR よりも多少ではあるが効率が良いことが示されている [1] [8] [10]．しかし一方において Halling らの研究では CBR が PBR よりも効率が良いという結果が示されている [4]．またいくつかの文献においてはこれらの手法の間には優位な差がないという結果も示されている [3] [7]．

従来の研究において実験結果に幅が見られるのは，人的要因がレビュー効率に与える影響がレビュー手法の与える影響よりも大きいためだと考えられる．例えば，Thelin らは UBR と CBR の誤り発見率 (発見した誤りの数／誤りの総数) を比較している [10]．図1は Thelin らが2つの手法における誤り発見率を比較した結果を示したものであるが，さまざまな種類の誤りにおいて UBR のほうが CBR よりも優れていることが示されている．一方，それぞれのグラフにおいて点線で示されている

[*]Hidetake Uwano, 奈良先端科学技術大学院大学

[†]Masahide Nakamura, 奈良先端科学技術大学院大学

[‡]Akito Monden, 奈良先端科学技術大学院大学

[§]Ken-ichi Matsumoto, 奈良先端科学技術大学院大学

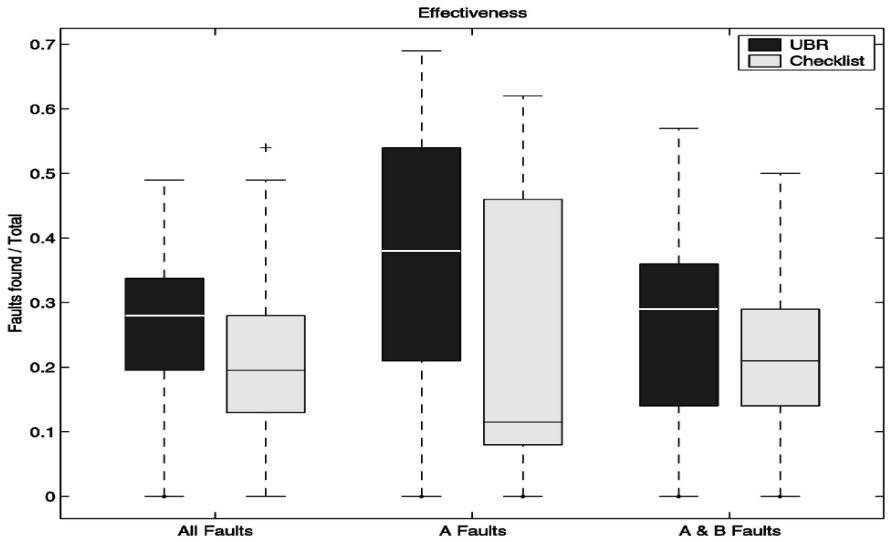

図 1　UBR と CBR の誤り発見率（文献 [10] から転載）

ように，同じ手法を用いた際の個人差は手法の差よりも大きく広がっている．ところが，このような個人ごとの性能の差については従来よく研究されていない．そこで本研究ではソフトウェアレビューの 1 つであるコードレビューにおける個人差を調べその要因を明らかにすることを目的とする．

コードレビューにおける個人差の要因を明らかにするために本研究ではレビュー作業者の視線の動きに着目する．一般に，ソースコードは新聞や雑誌といった通常の文章のようには読まれない．コードレビューでは，ある行における変数の値と他の行における値を見比べるために行から行へのジャンプや，プログラムの動作をシミュレーションするために，処理が行われる順に行を移動する動きが頻繁に発生する．このようなソースコードの読み方は作業者ごとに異なっており，レビュー性能に影響を与えていると予想される．そのため視線の動きを分析することでコードレビューにおける読み方の違いがレビュー効率に与える影響を調べることができると考えられる．

本実験では我々が構築したソフトウェアレビュー中の作業者の視線を計測・記録するための統合環境を用いた [12]．本論文ではソースコードをレビューしてもらう実験を行い，誤りを発見する際の視線の動きを分析した．その結果，被験者の視線の動きに特徴的なパターンが 3 つ見つかった．またソースコードを読み始めた直後に十分な時間をかけてコード全体を読んでいない場合，誤りを発見するまでの時間が延びる傾向があることが分かった．

本論文の構成は以下の通りである．2 章では視線を用いた関連研究について述べる．3 章では統合環境を用いて行ったレビュー実験について説明する．4 章では実験で得られたデータの分析結果について述べ，5 章ではまとめと今後の課題について述べる．

2　関連研究

視線の計測は初心者と熟練者の違いを分析することを目的に，特に認知科学の分野においてよく用いられている．Law らは腹腔鏡手術の訓練装置を使用している際の初心者と熟練者の視線の動きを分析している [6]．分析の結果，熟練者は初心者に

比べて患部をより集中して見ていることが明らかになっている．Kasarskis らはフライトシミュレータを使って飛行機の着地を行っている際のパイロットの視線を計測している [5]．その結果，熟練者が速度計を良く見ているのに対して初心者は高度計をより多く見ていることが明らかになっている．

デバッグやプログラム理解における視線を計測した研究も少数ではあるが行われている．Stein と Brennan はレビュー作業者に他の作業者がレビューした際の視線を見せることでバグ発見時間が短くなることを明らかにしている [9]．Torii らは被験者にソースコードとバグを含んだプログラムの動作結果を示し，デバッグを行う際の視線やキーストローク，発汗などを計測し，熟練者と初心者を比較した [11]．その結果，熟練者はデバッグ作業が進むに連れてバグが含まれていると思われる関数の数を限定していったのに対し，初心者は関数を限定することができなかった．Crosby と Stelovsky は誤りを含んだソースコードと正常に動作するプログラムを被験者に示し，ソースコードを理解する過程の視線を計測した [2]．その結果，初心者はコメントに視線が集中するのに対して，熟練者はコメントをあまり見ておらず，ソースコードを良く見ていることなどが明らかになった．

デバッグやプログラム理解における視線を計測したこれらの研究は本研究と関連が深い．しかし，どのようにプログラムを読み進めるべきかという知見は得られていない．一方，本研究では行単位でプログラマの視線を計測することで，プログラムを読み進める過程を細かく分析する．

3　実験

3.1　視線計測環境

我々はソフトウェアレビュー中の作業者の視線を計測するための統合環境を構築した [12]．本環境は非接触型視線計測装置である EMR-NC(nac 社製) と我々が開発したアプリケーション Crescent を用いることでレビュー作業者が現在注視しているドキュメントの行番号を時系列に記録する．また，記録した視線の動きをドキュメント上で再生することができるほか，視線の移動順序や各行に留まった時間を示すバーチャートを表示することができる．図 2 に Crescent の結果表示画面の例を示す．

3.2　レビュー対象

C 言語で書かれた 12 行から 23 行の 6 つのソースコードをレビュー対象とした．いずれのソースコードにもコメントは含まれておらず，1 つだけバグが埋め込まれている．それぞれのプログラムについて短く単純で被験者が容易に理解し記憶できる仕様書を用意した．表 1 に各プログラムの仕様と埋め込まれたバグについて示す．

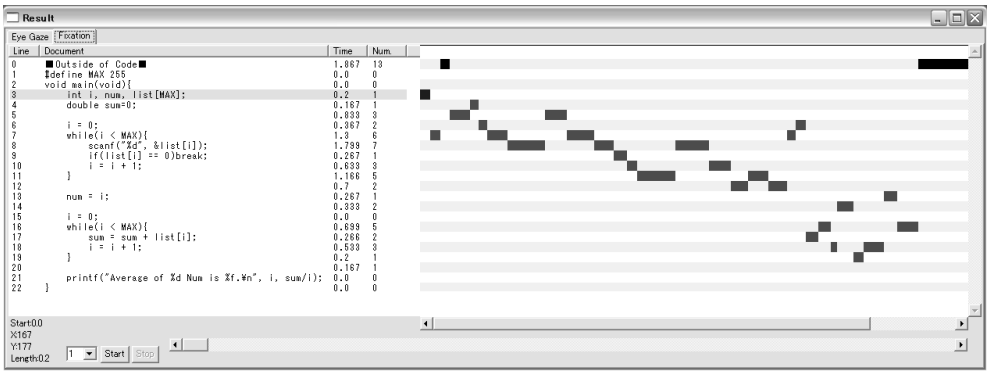

図 2　結果表示画面の例

表 1　実験で使用されたプログラムの仕様と混入されたバグ

プログラム	行数	プログラムの仕様	混入したバグ
Sum-5	12	ユーザから 5 つの整数を受け付け，その合計を返す.	合計値が代入される変数が初期化されていない.
Accumulate	20	0 以上の値 n を受け付け，1 から n までの合計値を返す.	while 文の条件式が $(i <= n)$ ではなく $(i < n)$ となっている.
Average-5	16	ユーザから 5 つの整数を受け付け，その平均を返す.	変数のキャストが行われておらず，誤差が生じる.
Average-any	22	ユーザから 255 個以下の任意の数の整数値を受け付け，その平均を返す.	while 文の条件式が誤っており，常に 255 個入力されたものとして計算する.
Swap	23	ユーザから 2 つの数値を受け付け，swap 関数を使ってそれらを入れ替え出力する.	ポインタが正しく使われておらず，2 つの数値が入れ替えられずに出力される.
Prime	18	整数 n を受け付け，その値が素数かどうかを判定する.	if 文の条件式が誤っており，正解と逆の結果を出力する.

ソースコードは 20 ポイントのフォントで表示され，Crescent 上でスクロールをすることなく全ての行を見ることができるようにした.

3.3　実験手順

奈良先端科学技術大学院大学情報科学研究科に在籍している博士前期課程の学生 5 名に表 1 で示した 6 つのソースコード全てに対してレビューを行ってもらい，埋め込まれたバグを探してもらった．被験者のプログラミング経験は 3 年から 4 年であり，全員が 1 度以上のレビュー経験があった．レビューの際，3.1 節で述べた環境を用いて被験者の視線の動きを記録した．それぞれのプログラムについてソースコードと仕様書が被験者に与えられた．被験者はレビュー開始前およびレビュー中，C 言語の仕様とプログラムの仕様について質問することができた．レビュー方法は AHR とし，チェックリストや特定の役割などは与えなかった．被験者が 1 つのソースコードをレビューする過程（1 回のタスク）は以下の通りである.

1. 被験者の視線を正確に計測するために視線計測装置のキャリブレーション（調整）を行う.
2. 被験者に対してプログラムの仕様を口頭で伝える.
3. 被験者がレビューを開始すると同時に視線の計測を開始する.
4. 被験者がバグを見つけたら一度作業を中断し，見つけたバグについて口頭で説明してもらう.
5. 被験者の見つけたバグが正しければレビューを終了する．誤っていた場合，ステップ 3 に戻りレビューを再開する．レビューはバグを発見するか，レビュー時間が 5 分を超えるまで行う.

3.4　実験結果

実験の結果得られたデータから被験者の瞬きや，よそ見などにより視線データを取得できなかった際のデータと測定エラーによる無効なデータを除外した．また，1 回のタスクの中で無効なデータが全体の 30 %を越えたものについては分析から除外した．被験者 5 名に対してのべ 30 回のレビュータスクを行った結果，3 つのタスクを除外した 27 の分析対象データが得られた.

4 分析と考察

各被験者の視線について分析を行った結果，視線移動に 3 つのパターンが見つかった．本章ではこれらのパターンについて述べる．

4.1 スキャンパターン

実験において，レビュー開始時にコード全体を上から下に眺め，その後，特定の範囲を集中して読むという動作が良く見られた．本研究ではこのようなコード全体を眺める動作をスキャンと呼び，スキャン時の視線の動きをスキャンパターンと呼ぶ．視線情報を分析した結果，レビュー時間のうち初めの 30 %の間に平均してコード全体の 72.8 %を読んでいることがわかった．図 3 および，図 4 にスキャンパターンが良く見られた被験者の視線の動きを示す．これらの図の横軸は停留点（視線が一定時間留まった点）の番号を示しており，被験者がどのような順でソースコードの各行を注視したかを示している．図 3 で見られるように被験者 E はソースコード全体を 2 回スキャンした後にソースコードの中ほどに位置している while ループに集中している．図 4 では被験者 C はレビュー開始直後に 2 つある関数のヘッダー部分（1 行目と 13 行目）を読んでいる．その後，被験者は関数 makeSum()，main() をそれぞれ読み，その後は関数 makeSum() に集中している．

スキャンパターンはレビュー作業者の認知過程が表れているものと考えられる．すなわち，作業者はレビュー開始時にプログラム全体の構造を理解しようとしており，スキャン中にバグが含まれている可能性の高い位置を識別しているものと考えられる．したがってレビュー開始時に行われるスキャンの品質はレビューにおけるバグ発見効率に影響している可能性がある．

スキャンの品質とバグ発見効率の関係を調べるために各レビューにおけるスキャン時間とバグ発見時間を計測した．スキャン時間とはスキャン開始から空行を除いたソースコード全体の 80 %を読むのに費やされた時間，バグ発見時間はレビュー開始から埋め込まれたバグを発見するまでに費やした時間と定義した．バグ発見時間とスキャン時間の関係を図 5 に示す．この図からはスキャン時間とバグ発見時間

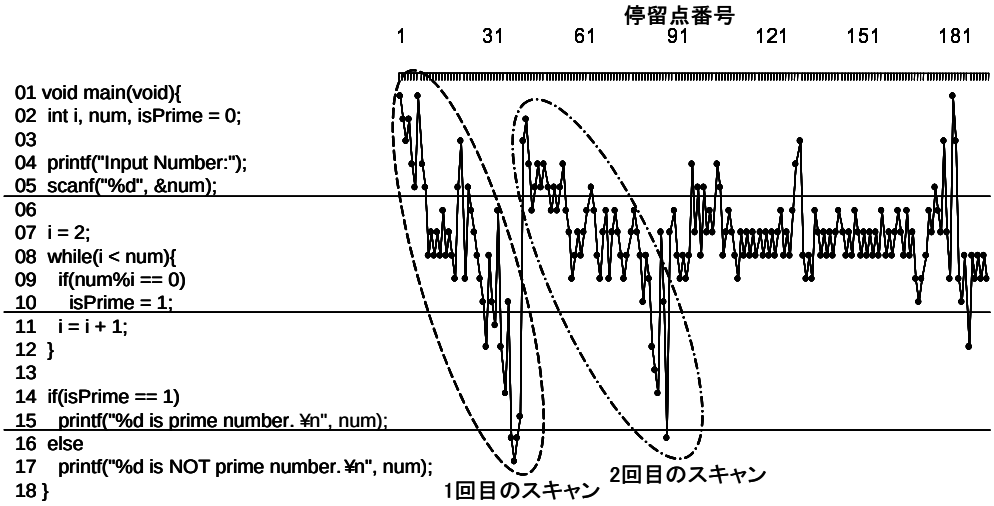

図 3　2 度のスキャンパターンを含む視線の動き（被験者 E：Prime）

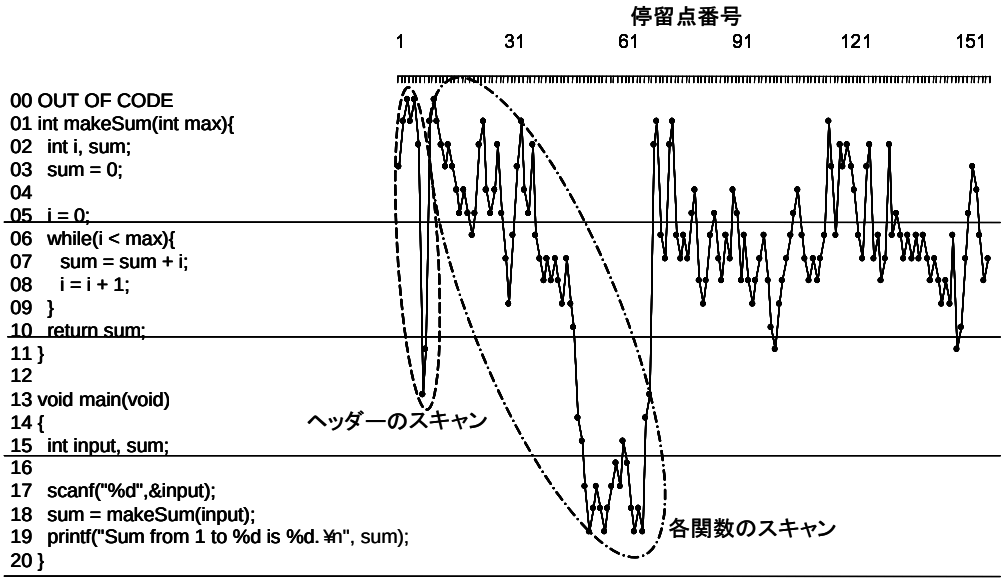

図 4　関数のスキャンを含む視線の動き（被験者 C：Accumulate）

の関係は見られなかった．これは被験者ごとにソースコードを読む速度が異なっているため，スキャン時間の長さがスキャンの質を表していないためだと考えられる．そこで，被験者ごとにスキャン時間とバグ発見時間を正規化（各被験者の平均のスキャン時間とバグ発見時間で割った値を算出）し，その関係を調べた．$n =$ 各被験者の実施したレビュータスク数 としたとき，被験者 s のタスク t_i における正規化スキャン時間 $s{-}scan(s, t_i)$ と正規化バグ発見時間 $s{-}detect(s, t_i)$ は以下の式で表される．

$$detect(s, t_i) = \text{被験者 } s \text{ のタスク } t_i \text{におけるバグ発見時間}$$
$$scan(s, t_i) = \text{被験者 } s \text{ のタスク } t_i \text{におけるスキャン時間}$$

$$s{-}detect(s, t_i) = \frac{n * detect(s, t_i)}{\sum_{k=1}^{n} detect(s, t_k)}$$

$$s{-}scan(s, t_i) = \frac{n * scan(s, t_i)}{\sum_{k=1}^{n} scan(s, t_k)}$$

図 6 に正規化したスキャン時間と正規化したバグ発見時間の関係を示す．この図で横軸は各レビューにおいて各被験者が被験者自身の平均と比べてどれだけ時間をかけてスキャンを行ったか，縦軸は各レビューにおいて各被験者が被験者自身の平均と比べてどの程度バグの発見に時間がかかったかを示している．また最小二乗法による 3 次の近似線もあわせて示す．図 6 はスキャン時間が平均よりも短いとき，バグ発見までの時間が延びる傾向があることを示している．本実験ではスキャン時間

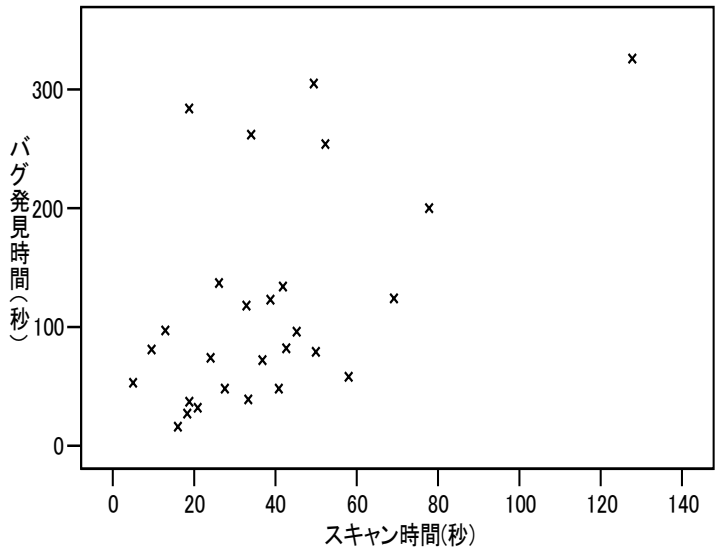

図 5　スキャン時間とバグ発見時間の関係

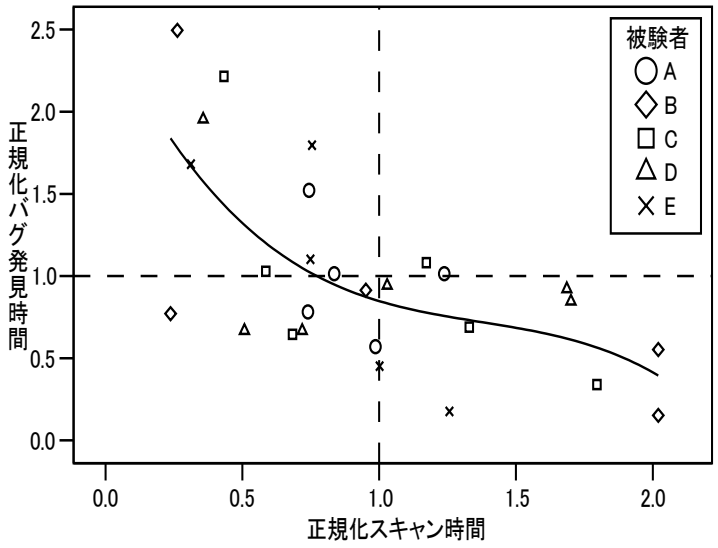

図 6　正規化したスキャン時間と正規化したバグ発見時間の関係

が 1.0 以下のときバグ発見時間は最大で 2.5 倍になっている．一方で，スキャン時間が 1.0 以上のときほぼ全てのタスクでバグ発見時間が平均以下になっている．また，近似線は，スキャン時間が平均より短いほどバグ発見時間が長くなること，スキャン時間が平均より長いほどバグ発見時間が短くなることを示している．

この結果はスキャンにより多くの時間をかけることでバグを発見するまでの時間を短くすることができることを示している．これは以下のように解釈することができる．ソースコードを注意深くスキャンするレビュー作業者はスキャン中によりプ

ログラムの構造を正しく理解し，バグが埋まっていると思われる行を識別することができる．図 6 の各被験者別の結果に着目すると，被験者 A はすべてのレビュー時において平均的な時間のスキャンを行っており，バグ発見時間が全被験者中最も短く，安定していた．一方，被験者 B および C はいくつかのタスクにおいてスキャン時間が各人の平均より著しく小さく，それらのタスクではバグ発見時間が各人の平均の 2 倍前後となっていた．一方，相対的にスキャン時間の大きなタスクでは，各人の平均以下の時間でバグを発見できていた．スキャン時間が相対的に小さかったタスクをより詳細に分析すると，バグとは無関係の行に集中している様子が見られた．以上のことから，スキャンを十分に行っていない場合，プログラムの構造を正確に把握することができずソースコード中の重要な行を見落としたり，レビューの効率を低下させる原因となっているのではないかと考えられる．

4.2 宣言文確認パターン

被験者の視線が，変数が最初に参照されている行に達したとき，短時間のうちにその変数が宣言されている行に戻る動きが見られた．このような視線の動きを宣言文確認パターンと呼ぶ．図 7 に宣言文確認パターンが見られる視線の動きを示す．この図では変数 i が始めて使用される 4 行目に到達したとき，変数が宣言されている 2 行目に 2 度戻る動きが見られる．同様の動きが 6 行目の input，7 行目の sum に到達した後にも見られる．全ての変数のうち 51.8 %において，始めて使用されたあと 3 秒以内に宣言文を確認する動きが確認された．このような視線の動きは被験者が変数の型を確認しようとする動作を反映していると考えられる．ただし，宣言文確認パターンについて，バグ発見効率との関連を分析したが，関係は見られなかった．

4.3 変数出現確認パターン

被験者の視線が，変数の出現した行に達したとき，短時間のうちにその変数が最近出現した行に戻る動きが見られた．このような視線の動きを変数参照確認パターンと呼ぶ．図 8 に変数参照確認パターンが見られる視線の動きを示す．この図で被験者が変数 ave を含む 15 行目に達したとき，最近 ave が出現した行である 14 行目に戻っている．そして 14 行目には変数 sum と i が含まれておりこれらを確認するために再び変数出現確認パターンが起こっている．被験者の視線には続いて while ブロック内の sum と i を含んだ行を確認する動作が見られる．このような視線の動きは被験者が変数の値を思い出す様子や，現在の変数の値を再計算する様子が反映されているものと考えられる．ただし，変数出現確認パターンについて，バグ発見

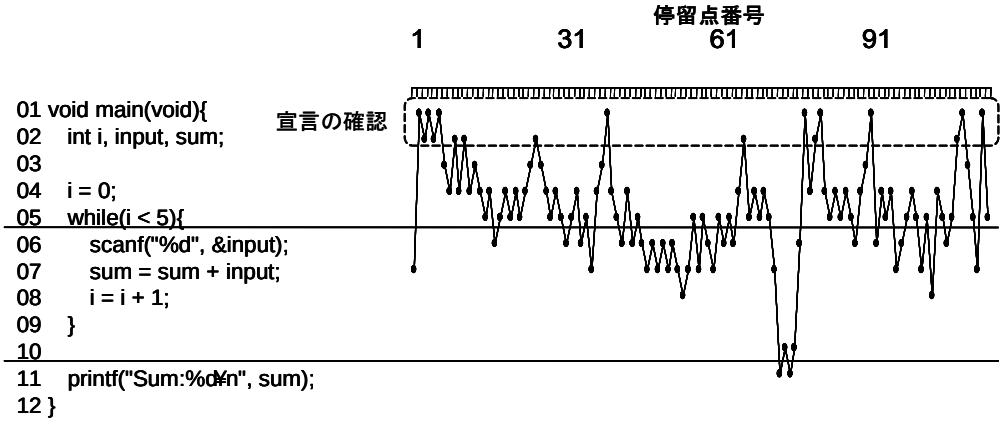

図 7　宣言文確認パターン（被験者 A：Sum-5）

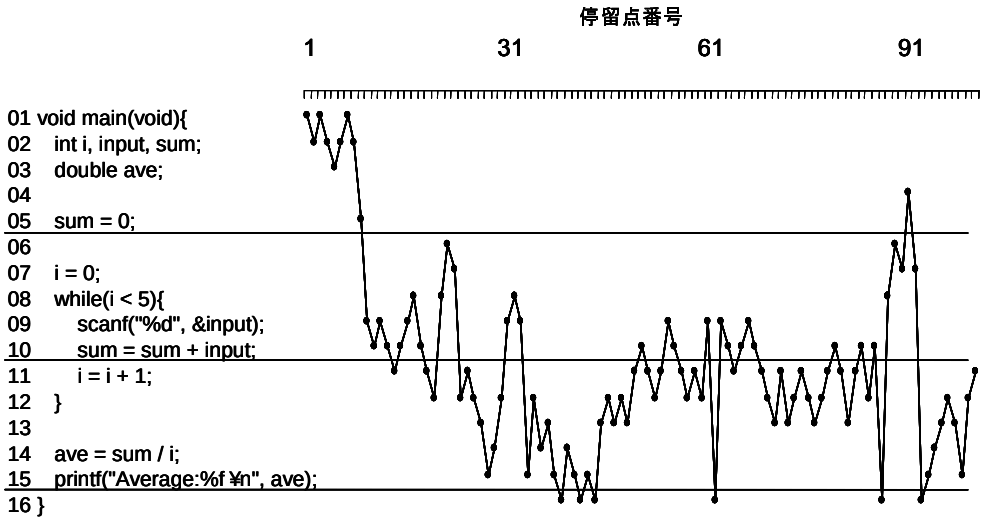

図 8　変数参照確認パターン（被験者 C：Average-5）

効率との関連を分析したが，関係は見られなかった．

5　まとめと今後の課題

本論文ではソフトウェアレビュー時における作業者の視線を計測する環境を用いて，コードレビューにおける個人差の要因を明らかにするための実験を行った．

実験では被験者の視線の動きからレビュー開始時にコード全体を眺めるというスキャンパターンが見つかった．視線データの分析から，レビュー開始時に十分なスキャンを行った被験者はコード内のバグを早く見つける傾向があることがわかった．これはレビュー開始時にコード全体を眺めることでプログラム全体の構造を把握し，個々の行の理解が容易になっているためと考えられる．

バグ発見効率との関連は見られなかったものの，分析の結果，スキャンパターン以外にも 2 つのパターン（宣言文確認パターン，変数出現確認パターン）を発見した．これらのパターンについてはレビュー効率との関係は見られなかったが，全ての被験者において出現していたため，作業者がプログラムを理解する過程において一般的に行われる行動であると考えられる．さらに分析を進めていくことで，レビュー作業者のプログラム理解過程がより詳細に明らかになれば，レビュー初心者の訓練や，プログラムの理解を助けるためのツールの開発に役立てることができると期待される．

今回の実験結果から，作業者の視線はコードレビューの様子を分析する手段として役立つことが分かった．被験者に作業中の思考について発言してもらう発話法や，作業中に定期的にインタビューを行うことで被験者の様子を観察する方法に比べ，視線を用いた観測は作業者の行動を妨げず，より自然な状態での記録が可能であり，レビューの観察には適した方法であるといえる．

本研究で構築した実験環境を用いることで要求仕様書や詳細設計書といった他のソフトウェアドキュメントのレビュー過程を分析することも可能である．これらのドキュメントをレビューする過程を実験により明らかにすることで，作業者の熟練度に応じた手法の提案や作業者の認知行動に即した支援ツールの開発が可能になると考えられる．今後の課題は，以下の項目を満たした，より現実に即した環境における作業者のレビュー過程を分析することである．

- より実際的な規模のソースコードを対象とする
- 事前にバグの数を知らされていない状態で，複数のバグを含んだソースコードをレビューする
- チェックリストやレビューにおける視点を与えた状態でレビューする

謝辞 本研究の一部は、文部科学省「e-Society 基盤ソフトウェアの総合開発」の委託に基づいて行われた.

参考文献

[1] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, and M. V. Zelkowitz, " The empirical investigation of perspective-based reading," Empirical Software Engineering: An International Journal, Vol. 1, No. 2, pp. 133-163, 1996.
[2] M. E. Crosby, and J. Stelovsky, "How do we read algorithms? A case study," IEEE Computer, Vol. 23, No. 1, pp. 24-35, 1990.
[3] P. Fusaro, F. Lanubile, and G. Visaggio, "A replicated experiment to assess requirements inspection techniques, " Empirical Software Engineering: An International Journal, Vol. 2, No. 1, pp. 39-57, 1997.
[4] M. Halling, S. Biffl, T. Grechenig, and M. Kohle, "Using reading techniques to focus inspection performance, " In Proceedings of 27th Euromicro Workshop Software Process and Product Im-provement, pp. 248-257, 2001.
[5] P. Kasarskis, J. Stehwien, J. Hichox, A. Aretz, and C. Wickens, "Comparison of expert and novice scan behaviors during VFR flight," In Proceedings of the 11th International Symposium on Aviation Psychology, 2001.
[6] 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.
[7] J. Miller, M. Wood, M. Roper, and A. Brooks, "Further experiences with scenarios and check-lists," Empirical Software Engineering: An International Journal, Vol. 3, No. 3, pp. 37-64, 1998.
[8] A. A. Porter, and L. Votta, "Comparing detection methods for software requirements inspection: A replication using professional subjects," Empirical Software Engineering: An International Journal, Vol. 3, No. 4, pp. 355-380, 1998.
[9] R. Stein, and S. E. Brennan, "Another person's eye gaze as a cue in solving programming problems," In Proceedings of the 6th International Conference on Multimodal Interface, pp. 9-15, 2004.
[10] T. Thelin, P. Runeson, and C. Wohlin, "An experimental comparison of usage-based and check-list-based reading," IEEE Transaction on Software Engineering, Vol. 29, No. 8, pp. 687-704, 2003.
[11] K. Torii, K. Matsumoto, K. Nakakoji, Y. Takada, S. Takada, and K. Shima, "Ginger2: An envi-ronment for computer-aided empirical software engineering," IEEE Transactions on Soft-ware Engineering, Vol. 25, No. 4, pp. 474-492, 1999.
[12] 上野 秀剛, 中道 上, 井垣 宏, 門田 暁人, 中村 匡秀, 松本 健一, " プログラマの視線を用いたレビュープロセスの分析", 電子情報通信学会技術研究報告 SS2005-15, pp.21-26, June 2005.