NAIST-IS-MT0451019

修士論文

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

上野 秀剛

2006 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。

上野秀剛

審査委員：

松本 健一	教授	(主指導教員)
西谷 紘一	教授	(副指導教員)
門田 暁人	助教授	(副指導教員)

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*

上野秀剛

内容梗概

本研究の目的は，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いて人的要因がレビュー性能に与える影響を明らかにすることである．本研究ではレビュー対象物を精読する際の視線を計測するために，視線計測装置を含んだハードウェアコンポーネントと，ソフトウェアツール Crescent からなる統合環境を構築した．この環境はレビュー作業者がソースコードのどの行を見ているかを時系列に記録・再生できる．

構築した環境を用いて，コードレビュー時の視線とレビュー効率の関係を分析するための実験を行った．その結果，作業者の視線に（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）ある変数が初めて使われたときに変数宣言部を確認する動作，（３）ある変数が現れたとき，変数が直前に使用され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．また，レビュー開始時に（１）の動作を十分に行っていない被験者は，誤り検出までの時間が長くなる傾向にあることが視線データの分析からわかった．

キーワード

レビュー，ソースコード，誤り検出，視線，注視点

*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-ISMT0451019, 2006 年2 月2 日.

Factor Analysis of Code Review Performance Using Programmers' Eye Movement *

Hidetake Uwano

Abstract

The goal of this study is to analyze the performance of individuals in reviewing source code of computer program. In this study, we constructed integrated environment to measure the eye movements of the code reviewers. The environment consists of hardware components including a non-contact eye camera, and a software application Crescent. On the environment, the line number of the source code that reviewer is currently looking at is recorded.

Using the environment, we conducted an experiment to analyze a correlation of reviewer's eye movement and review efficiency. As a result of the experiments, we found three patterns in the subject's eye movements; 1) Subjects scan the whole lines of the source code from the top to the bottom briefly, 2) When subject reaches a code line where a variable is firstly used, the subject retraces declaration line of the variables, and 3) When subject reaches a code line where a variable is used, the subject retraces the lines where the variable has been recently referred.

Quantitative analysis showed that reviewers who did not spend enough time for the scanning of source code tend to take more time for finding defects.

Keywords:

Review, Source Code, Defect Detection, Eye Movement, Gazing Point

---

.*Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT0451019, February 2, 2006.

目次

1. はじめに..1

2. 関連研究..4

3. コードレビューにおける視線計測 6
3.1. 要求...6
3.2. 視線計測環境 7
3.2.1. ハードウェアコンポーネント 7
3.2.2. ソフトウェアコンポーネント：Crescent.......................................8

4. 実験 12
4.1. 被験者 12
4.2. レビュー対象 12
4.3. 実験手順...12
4.4. 実験データの正当性...14

5. 実験結果の分析 15
5.1. スキャンパターン 15
5.2. スキャンパターンとレビュー性能の関係 17

6. 考察 19
6.1. 視線の動きとレビュー作業者の思考...19
6.2. 他の視線パターン 20
6.2.1. 宣言文確認パターン...20
6.2.2. 変数参照確認パターン 21

7. まとめと今後の課題..23

謝辞..25

参考文献 26

付録 29
A.　プログラム Sum-5 29
B.　プログラム Accumulate 30
C.　プログラム Average-5 31
D.　プログラム Average-any 32
E.　プログラム Swap 33
F.　プログラム Prime 34

図目次

図 1 UBR と CBR の誤り発見率（文献[21]から転載） 2
図 2 Crescent の構成図 9
図 3 ソースコード表示用ウィンドウ 10
図 4 結果表示ウィンドウ 11
図 5 スキャンパターンを含む視線（被験者 E : Prime） 16
図 6 スキャンパターンを含む視線（被験者 C : Accumulate） 16
図 7 初回スキャンとバグ発見時間の相関 18
図 8 宣言文確認パターン（被験者 A : Sum-5） 21
図 9 変数参照確認パターン（被験者 C : Average-5） 22

表目次

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 13
表 2 インタビューで得られたコメント 20

1. はじめに

ソースコードレビュー（以後，単にコードレビューとする）とはコンピュータ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，コードレビューは人間のレビュアー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．文献[26]ではレビューはソフトウェア成果物内の50％から70％の誤りを発見することができるとされており，その効果は非常に高い．

従来，ソフトウェア工学の研究分野において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ている．それらの手法はいずれも，作業者にある特定の基準を定め，ドキュメントを読むように定義されている．Checklist-Based Reading(CBR)とはコード中に良く現れる誤りについてのチェックリストを作業者に提示し，それを元にレビューを行う手法で，ソフトウェア開発現場では最も広く使用されている[5]．Perspective-Based Reading(PBR)とは設計者，プログラマ，テスタのようなさまざまな視点を与えられ，その視点からレビューを行う手法である[16]．Defect-Based Reading(DBR)では特定の種類の誤りに着目してレビューを行う[14]．Usage-Based Reading(UBR)ではユーザの視点からレビューを行う[20]．一方で，特定の規準を与えずにレビューを行う方法をAd-Hoc Reading(AHR)と呼ぶ．

これらの手法の性能を比較するため，今までに多くの実験が行われている[3]．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．いくつかの論文において，CBRはAHRと同程度の効率しかないことが確認され，UBR，PBR，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][13][14][16][21]．しかし一方において文献[7]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．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[6][9][11][15].

従来の研究においてレビュー効率の評価実験結果に幅が見られるのは，人的要因がレビュー結果に与える影響が大きく，また個人の性能差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，文献[21]では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． 図 1は2つの手法における誤り発見率を比較したものであるが， 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．一方で，それぞれのグラフにおいて点線で示されているように，同じ手法を用いている際の個人差は手法の差よりも大きく広がっている．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．本研究では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[23][24][25].

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

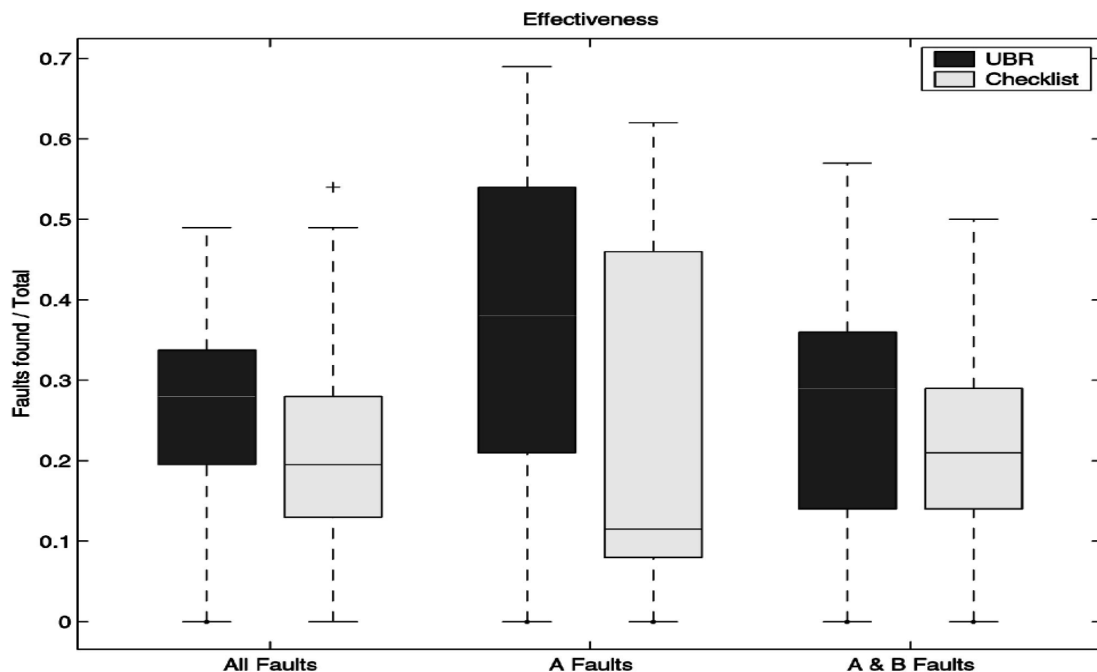

図 1 UBRとCBRの誤り発見率（文献[21]から転載）

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ような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いると予想される．そのため視線の動きを計測・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．

本研究ではコードレビュー中の作業者の視線を計測･記録するための環境を構築した．非接触型視線計測装置であるEMR-NC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計測できる．Crescentはある行から他の行へと移動するような行を基準とした視線の動きと，各行を注視した時間の長さを記録でき，また，記録した視線の動きを再生することができる．

構築した環境を用いて5人の被験者を対象に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章では開発した環境について説明し，4，5章では実験とその分析結果について述べる．6章で分析結果の考察を行い，7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[10]．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[8]．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．

ソフトウェア工学の分野においても視線移動を用いた研究がいくつか行われている．Zhaiらは視線計測装置をポインティングデバイスとして用いるための手法を提案しており，視線を用いることでマウスなど従来のポインティングデバイスよりも疲労が少なくなることが示されている[27]．NakamichiらはWebページのユーザビリティを評価するために視線を用いた実験を行っている[12]．実験の結果，実験中に記録した視線データを被験者に見せることで，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[18]．

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．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[22]．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[4]．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の関連が深い．しかし，どちらの場合でも被験者にプログラムの動作を見せているのに対し，本研究が対象とするコードレビューが行われる段階では動作するプ

ログラムが存在しないことが多く，これらの結果を適用することはできない．また，デバッグやプログラム理解において良く使われるデバッガを使用していないなど問題点が多い．

3. コードレビューにおける視線計測
3.1. 要求

この章では，レビュー中の視線からレビュー活動の特徴を明らかにするために実験環境に要求される特性について述べる．これらの要求を満たすことでコードレビューにおける人的要因を分析するのに適した実験環境を構築することができる．

要求1：行単位の視線追跡

プログラムは文（Statement）で構成されており，多くのプログラムにおいて1つの文は1行に書かれている．したがってレビュー作業者はプログラムを理解する際に1行を単位として読むと考えられる．このため構築する環境では作業者が現在どの行を見ているのかを識別しなければならない．このとき，画面上に表示されたコードのうちの何行目を読んでいるかではなく，ソースコード全体の何行目（以後，論理行と呼ぶ）を読んでいるのかを識別する必要がある．論理行を識別するためにはソースコードの表示に用いられているフォントサイズやウィンドウの位置，ソースコードがどの程度スクロールされているかを元に計算する必要がある．

要求2：レビュー作業者の注目の識別

作業者の視線がある行に存在する場合，作業者が必ずしもその行を見ているとは限らない．実験環境では作業者がその行に注目したのか，行の上を通っただけなのかを識別する必要がある．そのためには視線が一定期間，ある行にとどまったかどうかを識別する必要がある．

要求3：時系列に基づいた視線移動の記録

作業者がどのような順序で各行を読んだのかを知ることはレビューの特性を分析するためには重要な手がかりとなる．同様に，各行に対する注視の長さからそれぞれの行に対する作業者の注意の度合いを知ることができる．したがって構築する環境では時系列に基づいた視線移動と各行への注視時間の合計の記録が必要である．

要求4：分析の支援

実験環境には，記録したデータを分析するための機能があることが望ましい．特に記録した視線移動の再生やデータの可視化といった機能は分析を容易にする．これらの機能は実験の際に補助的に行われるインタビューやレビュー初心者の教育のためにも有効である．

3.2. 視線計測環境

前節で述べた要求に基づいて，コードレビューにおける視線移動を計測するための環境を構築した．この環境は非接触型視線計測装置を含んだハードウェアコンポーネントと，コードレビュー時の視線移動を記録するためのソフトウェアコンポーネントCrescentからなる．この環境では作業者はPCモニター上に表示されたソースコードをレビューする．

3.2.1. ハードウェアコンポーネント

3.1節の要求1で述べたようにソースコードを注視する視線を行単位で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（http://www.nacinc.jp/）の非接触型視線計測装置，EMR-NCを採用した．EMR-NCの視線計測頻度は30回／秒であり，計測誤差はディスプレイ上の5.4ピクセルに相当する．これは20ポイントのフォントサイズでソースコードを表示したとき，0.25行に相当する．

ソースコードを表示するために21インチ液晶ディスプレイ（EIZO Flex-Scan771）を解像度1024×768に設定し使用した．また実験中に作業者の体が動くことにより発生する測定誤差を最小にするため，固定式の椅子を使用した．

EMR-NCにより計測された視線データはRS-232Cを経由してPCに送られる．受信したPCではEMR-NC付属のソフトウェアがCSVファイルにデータを保存する．保存されるデータはディスプレイ上の絶対座標値で表された視線位置とデータの取得時間である．また，付属ソフトウェアでは一定期間，一定の範囲にとど

まった視線である停留点（Fixation）を計算する．停留点は一般に作業者がその位置に注目していると解釈される．したがって，停留点を用いることによって要求2で述べた作業者の注視を識別することができる．保存される停留点のデータはディスプレイ上の絶対座標値で表された停留範囲の中心位置とデータ取得時間，一定の範囲に留まった長さである．

3.2.2. ソフトウェアコンポーネント：Crescent

3.2.1で説明したハードウェアコンポーネント上で動作するソフトウェアCrescent（Code Review Evaluation System by Capturing Eye movemeNT）を開発した．Crescentは視線計測装置が保存したデータを行単位のデータに変換する． CrescentはJavaとSWT(Standard Widget Tool)を用いて作成された約4000行のプログラムである．図 2に構築した環境とCrescentの構成を示す．以下では図 2を用いてCrescentが3.1節で述べた要求をどのように満たすかを説明する．

本研究では要求1で示したように視線計測装置が出力するディスプレイ上の絶対座標値をソースコードの論理行番号に変換する必要がある．Crescentの起動時，Outputモジュールが図 3に示すようなソースコード表示用ウィンドウを表示する．レビュー作業者はウィンドウ上に表示されたソースコードのレビューを行い，その際の視線の動きを視線計測装置が記録する．Crescentは視線計測装置が記録した絶対座標値で現された視線位置とソースコード表示用ウィンドウの絶対座標値を元にウィンドウ内における視線の相対座標値を求める．次にソースコードの表示に用いているフォントサイズと行間の幅からウィンドウ内の相対座標値を論理行番号に変換する．作業者はコードレビュー中，頻繁にソースコードのスクロールを行うため，ウィンドウ内の相対座標と論理行の対応も頻繁に変化する．そのため，Scrolling Captureモジュールが作業者のスクロール状態を監視し，相対座標値と論理行との対応を更新する．

要求2を満たすためにEMR-NC付属のソフトウェアが出力する停留点情報を用いた．停留点情報を上で示した手順によって論理行番号に変換することで，作業者がどの行にどの程度の期間注目していたかを識別できるようになる．

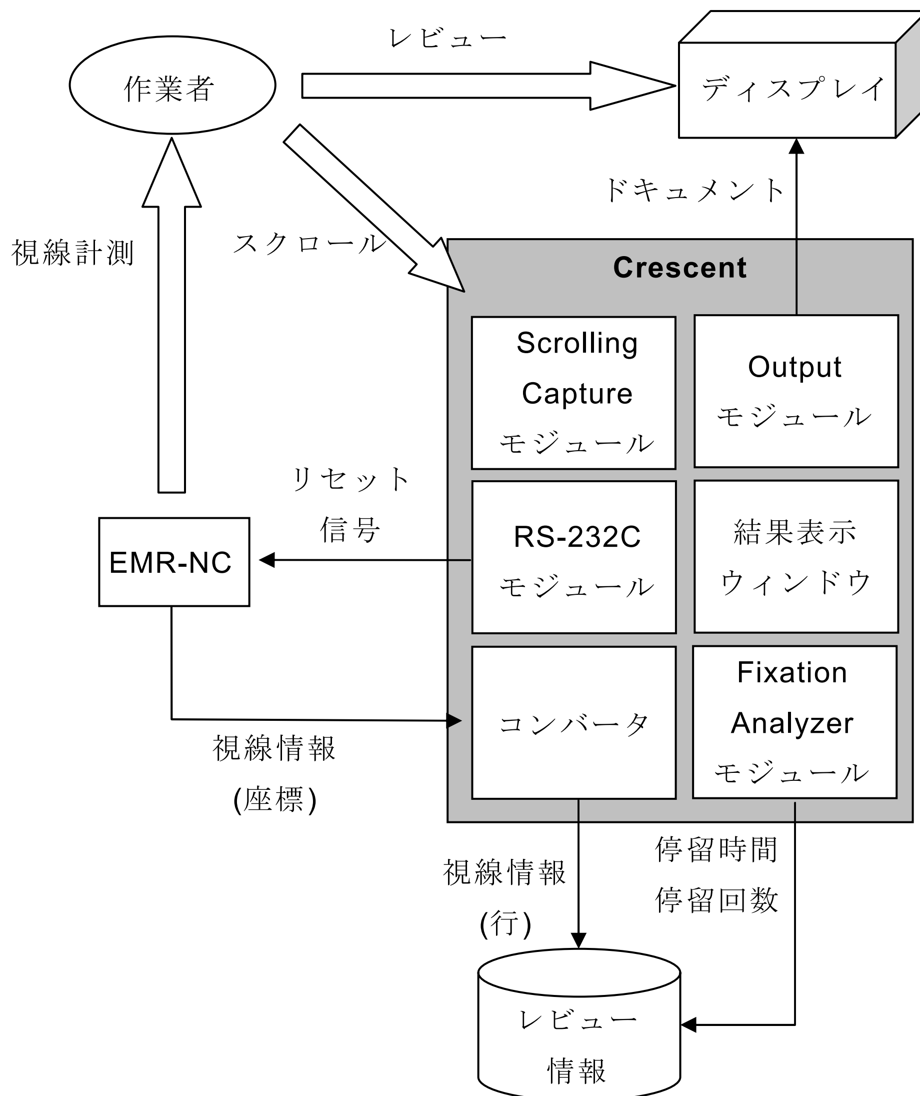

図 2Crescentの構成図

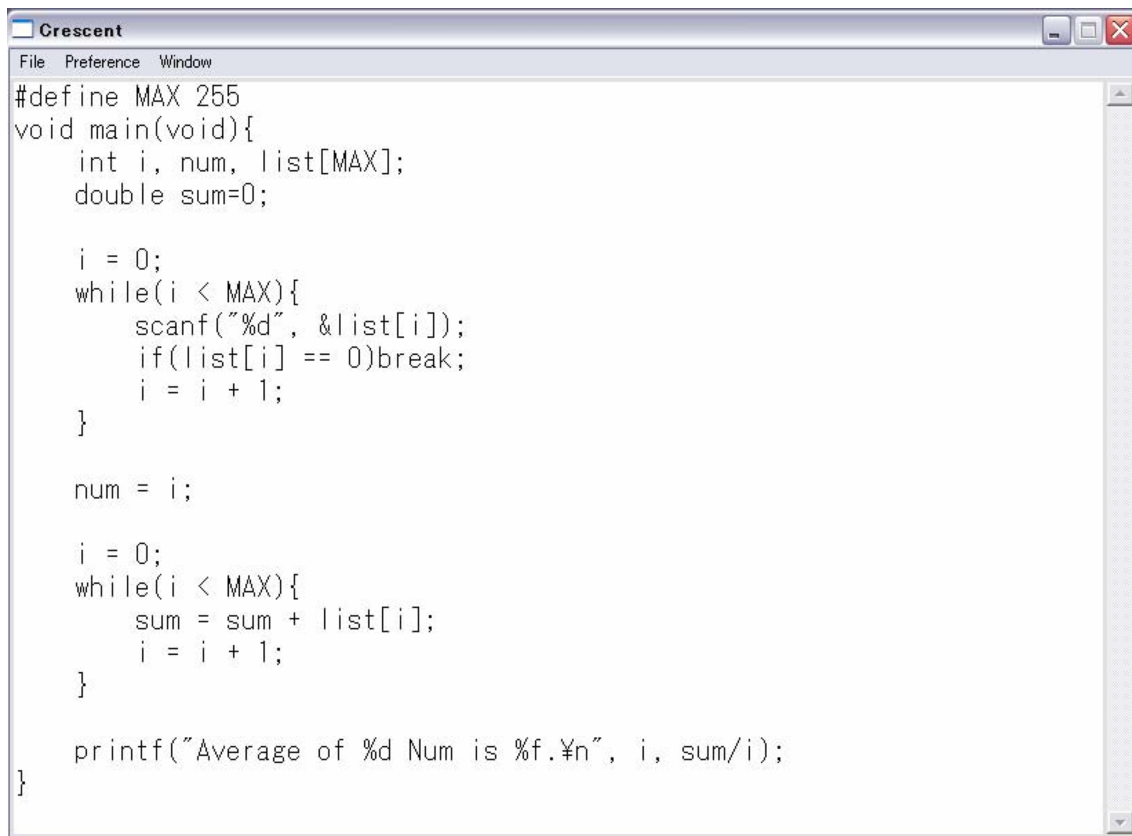

図 3ソースコード表示用ウィンドウ

要求3を満たすためにFixation Analyzerモジュールが停留点情報を元に各行に対する注視を時系列情報に変換する．この際，同一の行に対する連続した複数の停留点は1行に対する注目としてまとめられる．また各行ごとに注視した時間の合計と注視回数を計算している．

要求4を満たすため，Crescentは結果表示ウィンドウを備えている．図 4に結果表示ウィンドウの例を示す．このウィンドウはレビューが行われたソースコードと作業者の視線の動きをバーチャートで示したものを並べて表示する．また視線の時系列情報に合わせてコードと対応するバーをハイライトすることによって視線の再生が可能である．このとき，任意の時間から再生を行うことや，スロー再生が可能である．また，他のツールを用いた分析が行えるように，行単位の時系列情報をCSVファイルに出力することができる．

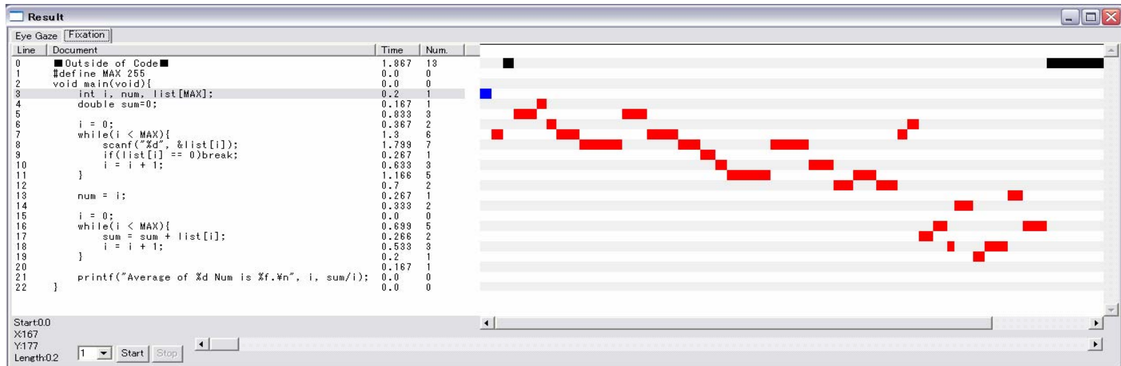

図 4結果表示ウィンドウ

CrescentはRS-232Cを経由して視線計測装置にリセット信号を送ることができる．リセット信号を送ることによりレビューの開始・終了と視線計測装置の計測開始・終了時間の同期を取ることができ，ソースコードのスクロールイベントの時間と視線情報の時間のずれを最小にしている．

4. 実験

本章では構築した環境を用いて行ったコードレビュープロセスの観察実験について説明する．

4.1. 被験者

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過程の学生5名に実験に参加してもらった．被験者のプログラミング経験は3年から4年であった．またいずれの被験者も最低1度のレビュー経験があった．

4.2. レビュー対象

C言語で書かれた12行から23行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にプログラムの仕様と埋め込まれたバグについて示す．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．

4.3. 実験手順

被験者に4.2節で示したソースコードに対して個別に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．レビューの際，3.2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視点などは与えなかった．

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

プログラム名	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までの整数の合計値を返す．	while 文の条件式が(i <= n)でなければいけないのに(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．	変数のキャストが正しく行われておらず，平均の計算に誤差が生じる．
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いるため，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，正しい答えとは逆の結果を出力する．

被験者が1つのソースコードをレビューする過程は以下の通りである．

1. 被験者の視線の動き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とスクロール状態の記録を開始する．
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．

本研究では直径30ピクセルの円中に50ms以上留まっていた視線を停留点とした．

4.4. 実験データの正当性

レビュー終了後，取得した視線データの正当性を確認した．被験者の瞬き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破棄した．また1回のタスクの中で破棄データが全体の30%を越えたものについては分析から除外した．6プログラム5名で計30回行ったうち，3つのタスクが分析から除外された．

5. 実験結果の分析

実験結果について，結果表示ウィンドウが可視化したレビュー中の視線情報と，Crescentが出力した時系列情報を用いて分析を行った．図 5にCrescentが出力した時系列情報をグラフにしたものを示す．図 5は被験者EがソースコードPrimeをレビューした際の視線の動きを示している．各被験者の視線について分析を行った結果，視線移動に特定のパターンが見つかった．

5.1. 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．視線情報を分析した結果，平均してレビュー時間のうち初めの30%の間にコード全体の72.8%を読んでいることがわかった．このような全体を眺める動作をスキャンと呼ぶ．

図 5および，図 6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 5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 6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．

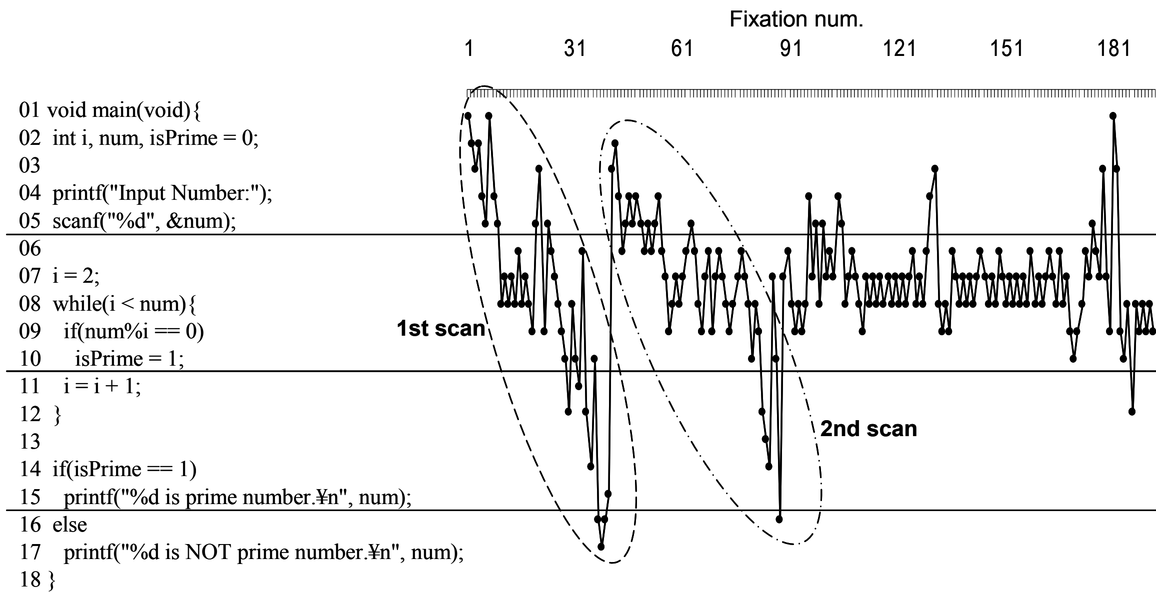

図 5スキャンパターンを含む視線（被験者E：Prime）

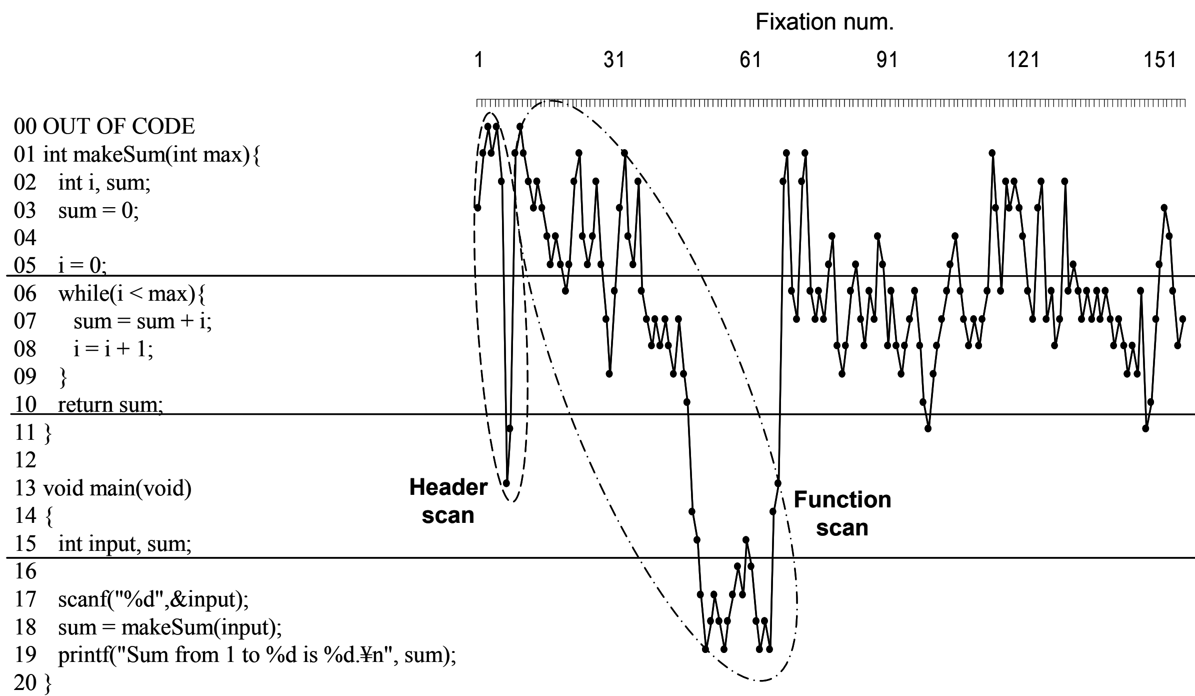

図 6 スキャンパターンを含む視線（被験者C：Accumulate）

5.2. スキャンパターンとレビュー性能の関係

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と考えられる．

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で定義される．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の時間と定義した．スキャン時間がスキャンの品質を表すという仮定の元，バグ発見時間とスキャン時間の関係を分析した．

図 7に各レビュー作業におけるスキャン時間とバグ発見時間の相関を示す．この図では横軸はスキャン時間を縦軸はバグ発見時間を示している．どちらの値もそれぞれの平均値で正規化している．図 7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．特にスキャン時間が0.8以下のときバグ発見時間が最大で2.5倍になっている．一方で，スキャン時間が0.8以上のときバグ発見時間は平均以下になっている．

この結果はソースコードをより長い時間をかけてスキャンをすることがより効率の良いバグ発見につながってい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．一方，スキャンを十分に行っていない作業者は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，バグとは関係のない行に集中したりするため，レビューの効率を落としている．

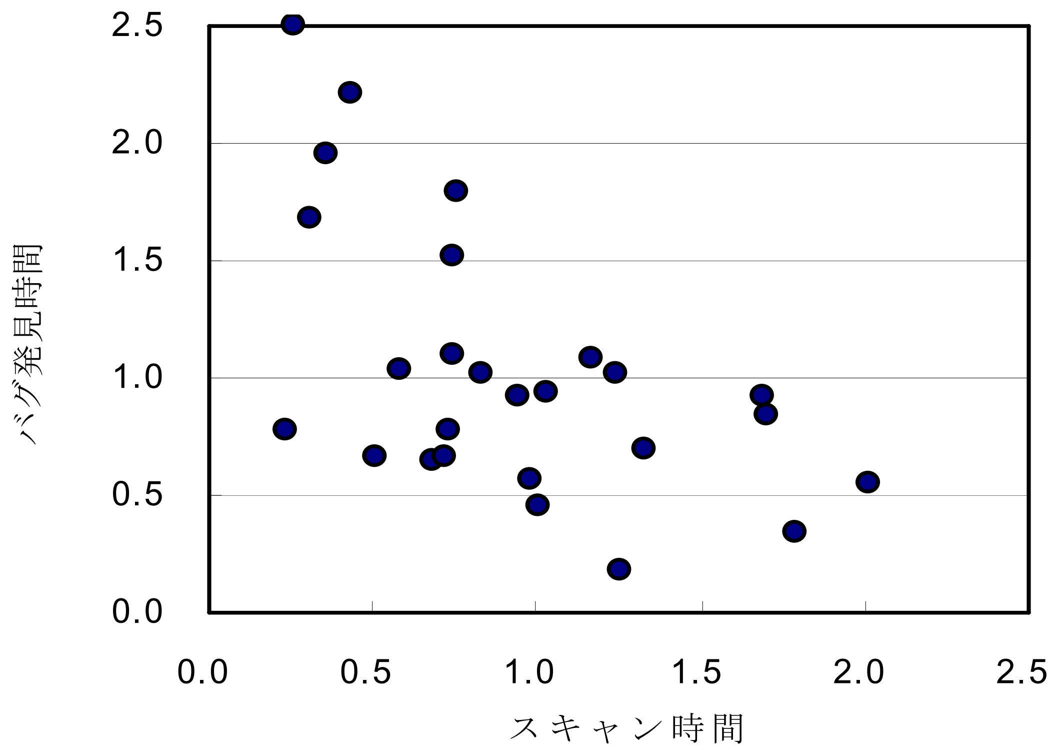

図 7 初回スキャンとバグ発見時間の相関

6. 考察

本章では今回の実験で観測された，定量的な差は出なかったが，興味深い発見について述べる．

6.1. 視線の動きとレビュー作業者の思考

各被験者にレビューを行ってもらった後，視線の動きが何を反映しているのかを調べるために2種類のインタビューを被験者に対して行った．

1回目のインタビューでは被験者にソースコードを見せ，レビュー中に何を考えていたのかを尋ねた．その結果，ほとんどの被験者は全体的な，あるいは抽象的なレビューの方針やソースコードの理解戦略，レビューの流れなどについて述べていた．表 2の1列目に1回目のインタビューで得られたコメントの例を示す．

2回目のインタビューではソースコードと同時にCrescentの結果表示ウィンドウで視線の動きを見せながら1回目と同様の質問を行った．2回目のインタビューでは1回目のときよりも細かく，ソースコードに関連したコメントを得ることができた．表 2の2列目に2回目のインタビューで得られたコメントの例を示す．表2に示されるようになぜ作業者がある特定の行に注目したのか，あるいはなぜ注目しなかったのかについて聞くことができた．

この結果はレビュー中の視線の動きが作業者の思考に関する情報を多く含んでいることを示している．したがって視線情報を用いることによって，レビュー実験において従来用いられてきたインタビューの効果をより高めることができる．また，レビュー熟練者の視線を記録し初心者に見せることで，教育・訓練に用いることができると考えられる．

表 2インタビューで得られたコメント

1回目のインタビュー（ソースコードのみ）	2回目のインタビュー（ソースコードと視線の動き）
• 初めに main 関数を見てから他の関数を見た． • ２つめの while 文があやしいと思った． • 頭の中でプログラムをシミュレートした． • while 文のループ回数を確認した．	• ループの条件判定式は確認しなかった． • 変数の初期値を確認するために，変数宣言部を確認した． • 出力の処理は気にしなかった． • よく見る書き方をしていたので入力の処理は正しいと思った．

6.2. 他の視線パターン

実験ではスキャンパターン以外にも2つのパターンが見つかっている．この節ではこれらのパターンについて述べる．

6.2.1. 宣言文確認パターン

被験者の視線がある変数が最初に使わ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 8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．

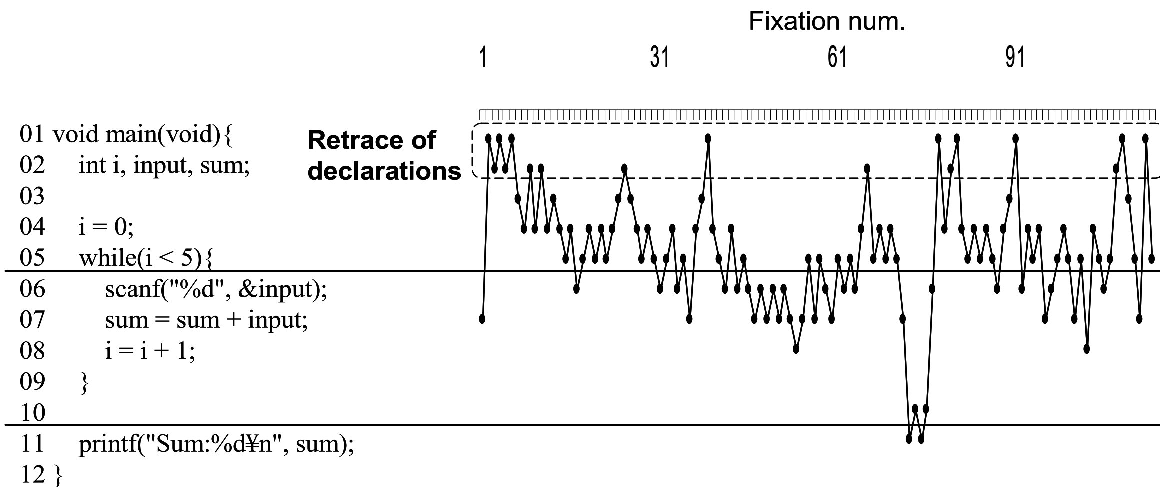

図 8 宣言文確認パターン（被験者A：Sum-5）

6.2.2. 変数参照確認パターン

このパターンは宣言文確認パターンに似ている．被験者の視線がある変数の使用されている行に達したとき，短時間のうちにその変数が最近使用され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 9 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が参照された行である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参照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．

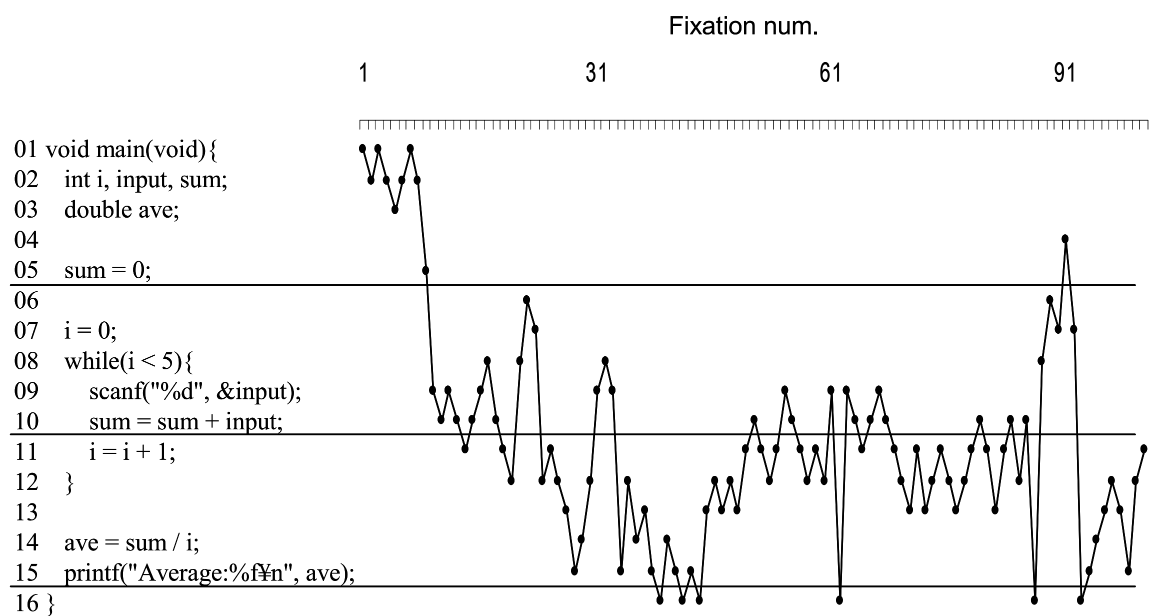

図 9 変数参照確認パターン（被験者C：Average-5）

7. まとめと今後の課題

本論文ではソースコードレビューにおけるレビュー作業者間の性能差を明らかにするために（1）レビュー中の視線を記録･再生するために非接触型視線計測装置を含んだハードウェアと開発したソフトウェアCrescentからなる実験環境の構築，および（2）構築した環境を用いて，ソースコードに埋め込まれたバグを探す実験を行った．

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．

統計的な差は見られなかったものの，実験結果にはスキャンパターン以外にも2つのパターンが見られた．これらのパターンからは作業者がプログラムを理解する過程を明らかにすることができるため，レビュー初心者の訓練に用いることができると考えられる．また，作業者のプログラムの理解を助けるためのツールの開発にも役立てることができると考えられる．

今回の実験結果から，作業者の視線からコードレビューの様子を分析することができることがわかった．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

本研究を進めるにあたり，多くの方々に御指導，御協力を賜りました．ここに謝意を添えて御名前を記させて頂きます．

奈良先端科学技術大学院大学 情報科学研究科，松本 健一 教授には，常日頃より御指導と御助言を頂きました．また，日常生活についても御配慮，御助言を頂きました．心より感謝申し上げます．

本研究を直接指導して頂いた，奈良先端科学技術大学院大学 情報科学研究科，門田 暁人 助教授には，常日頃より熱心な御指導を頂き，実験のデザイン，実験結果の分析について適切な御助言を頂きました．心より感謝申し上げます．

本研究を進めるにあたり，副指導教員を担当して頂いた，奈良先端科学技術大学院大学 情報科学研究科，西谷 紘一 教授には本研究について鋭い御指摘，御助言を頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科，中村 匡秀 助手には本研究について幅広い視点からの御意見，御指摘を頂きました．また論文作成においても熱心な御指導を頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科，大平 雅雄 助手には常日頃から研究についての適切な御助言を頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科，井垣 宏 特任助手には実験についてさまざまな御意見，御指摘を頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科，中道 上 様には視線計測装置の扱い方や実験について貴重な御意見，御指摘を頂きました．心より感謝申し上げます．

本研究を進めるにあたり，実験の被験者として参加していただいた方々に心より感謝申し上げます．

最後に，御助言や御協力を頂いたソフトウェア工学講座の皆様に，心より感謝申し上げます．

参考文献

[1] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgärd, 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] B. W. Boehm, "Software Engineering Economics," Prentice Hall, 1981.

[3] M. Ciolkowski, O. Laitenberger, D. Rombach, F. Shull, and D. Perry, "Software inspection, reviews & walkthroughs," In Proceedings of the International Conference on Software Engineering (ICSE), pp. 641-642, 2002.

[4] M. E. Crosby, and J. Stelovsky, "How do we read algorithms? A case study," IEEE Computer, Vol. 23, No. 1, pp. 24-35, 1990.

[5] M. E. Fagan, "Design and code inspection to reduce errors in program development," IBM Systems Journal, Vol. 15, No. 3, pp. 182-211, 1976.

[6] 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.

[7] 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 Improvement, pp. 248-257, 2001.

[8] 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.

[9] F. Lanubile, G. Visaggio, "Evaluating defect detection techniques for software requirements inspections," ISERN Technical Report-00-08, 2000.

[10] 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.

[11] J. Miller, M. Wood, M. Roper, and A. Brooks, "Further experiences with scenarios and checklists," Empirical Software Engineering: An International Journal, Vol. 3, No. 3, pp. 37-64, 1998.

[12] N. Nakamichi, M. Sakai, J. Hu, K. Shima, M. Nakamura, and K. Matsumoto, "WebTracer: Evalu-

ating web usability with browsing history and eye movement," In Proceedings of 10th International Conference on Human-Computer Interaction (HCI International 2003), pp. 813-817, 2003.

[13] 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.

[14] A. A. Porter, L. G. Votta, and V. R. Basili, "Comparing detection methods for software requirements inspection ― A replicated experiment," IEEE Transaction on Software Engineering, Vol. 21, No. 6, pp. 563-575, 1995.

[15] K. Sandahl, O. Blomkvist, J. Karlsonn, C. Krysander, M. Lindvall, and N. Ohlsson, "An extended replication of an experiment for assessing methods for software requirements inspections," Empirical Software Engineering: An International Journal, Vol. 3, No. 4, pp. 281-406, 1998.

[16] F. J. Shull, "Developing Techniques for Using Software Documents: A Series of Empirical Studies," PhD thesis, Univ. of Maryland, 1998.

[17] F. Shull, I. Rus, and V. Basili, "How perspective-based reading can improve requirements inspections," IEEE Computer, Vol. 33, No. 7, pp. 73-79, 2000.

[18] 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.

[19] T. Thelin, C. Andersson, P. Runeson, and N. Dzamashvili-Fogelström, "A replicated experiment of usage-based and checklist-based reading," In Proceedings of 10th IEEE International Symposium on Software Metrics (METRICS'04), pp. 246-256, 2004.

[20] 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.

[21] 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.

[22] K. Torii, K. Matsumoto, K. Nakakoji, Y. Takada, S. Takada, and K. Shima, "Ginger2: An environment for computer-aided empirical software engineering," IEEE Transactions on Software Engineering, Vol. 25, No. 4, pp. 474-492, 1999.

[23] H. Uwano, M. Nakamura, A. Monden and K. Matsumoto, "Analyzing individual performance of

source code review using reviewers' eye movement, " Eye Tracking Research and Applications 2006, (in Press).

[24] 上野 秀剛, 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バグ特定プロセスの分析", ソフトウェア信頼性研究会 第 2 回ワークショップ, June 2005.

[25] 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.

[26] Karl E. Weigers (著), ピアレビュー, 日経 BP ソフトプレス, 2004.

[27] 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.

付録

A. プログラムSum-5

```
void main(void){
      int i, input, sum;

      i = 0;
      while(i < 5){
            scanf("%d", &input);
            sum = sum + input;
            i = i + 1;
      }

      printf("合計値は%dです.¥n", sum);
}
```

- 仕様

整数の入力を5個受け付け，合計を返す.

- 混入されたバグ

sumが0クリアされていない.

B. プログラムAccumulate

```
int makeSum(int max){
    int i, sum;
    sum = 0;

    i = 0;
    while(i < max){
        sum = sum + i;
        i = i + 1;
    }
    return sum;
}

void main(void)
{
    int input, sum;

    scanf("%d",&input);
    sum = makeSum(input);
    printf("1から%dまでの合計は%dです.¥n", input, sum);
}
```

- 仕様

1から入力した数までを合計した値を出力する.

- 混入されたバグ

while文の終了条件で比較式が間違っている(i<max → i<= max)

C. プログラムAverage-5

```
void main(void){
      int i, input, sum;
      double ave;

      sum = 0;

      i = 0;
      while(i < 5){
            scanf("%d", &input);
            sum = sum + input;
            i = i + 1;
      }

      ave = sum / i;
      printf("平均値は%fです.¥n", ave);
}
```

- 仕様

整数の入力を5個受け付け，平均値を返す.

- 混入されたバグ

sum/iの結果がintであり，小数点が丸められる.

D. プログラムAverage-any

```
#define MAX 255
void main(void){
	int i, num, list[MAX];
	double sum=0;

	i = 0;
	while(i < MAX){
		scanf("%d", &list[i]);
		if(list[i] == 0)break;
		i = i + 1;
	}

	num = i;

	i = 0;
	while(i < MAX){
		sum = sum + list[i];
		i = i + 1;
	}

	printf("%d個の平均値は%fです.¥n", i, sum/i);
}
```

- 仕様

255個以下の整数を入力し，その平均値を返す.
0が入力されると入力が終了する（0は平均には含めない）.

- 混入されたバグ

合計を計算するループで終了条件が常に最大入力数（255）になっている.

E. プログラムSwap

```
void swap(int a, int b){
    int tmp;
    tmp = a; a = b; b = tmp;
}

void main(void){
    int list[2], i;

    i = 0;
    while(i < 2){
        printf("%dつ目:", i+1);
        scanf("%d", &list[i]);
        i = i + 1;
    }

    swap(list[0], list[1]);

    i = 0;
    while(i < 2){
        printf("交換後の%dつ目:%d¥n", i+1, list[i]);
        i = i + 1;
    }
}
```

- 仕様

2つの値を入力すると入れ替えて出力する.

- 混入されたバグ

swap関数へ参照渡しではなく値渡しをしている.

F. プログラムPrime

```
void main(void){
    int i, num, isPrime = 0;

    printf("判定したい整数を入力してください:");
    scanf("%d", &num);

    i = 2;
    while(i < num){
        if(num%i == 0)
            isPrime = 1;
        i = i + 1;
    }

    if(isPrime == 1)
        printf("%dは素数です.¥n", num);
    else
        printf("%dは素数ではありません.¥n", num);
}
```

- 仕様

入力された整数が素数かどうかを判定する.

- 混入されたバグ

isPrimeの値が反対に設定されている，またはif文の条件判定式が逆の結果になる.