システム創成工学専攻
情報システムコース
Department of Systems Innovation
Advanced Information System Course

令和４年度 専攻科特別研究論文

視線遷移データを用いたコードレビューの頻出パターン分析

Frequent Pattern Analysis of Code Review Using Gaze Transition Data

指導教員名　上野 秀剛 准教授
論文提出者名　二宮 凌真

独立行政法人 国立高等専門学校機構
奈良工業高等専門学校 専攻科
National Institute of Technology, Nara College
Faculty of Advanced Engineering

視線遷移データを用いたコードレビューの頻出パターン分析

Frequent Pattern Analysis of Code Review Using Gaze Transition Data

二宮 凌真
Ninomiya Ryouma

独立行政法人 国立高等専門学校機構
奈良工業高等専門学校 専攻科 システム創成工学専攻 情報システムコース
大和郡山市矢田町 22 番地（〒 639-1080）
National Institute of Technology, Nara College, Faculty of Advanced Engineering
22 Yata-cho, Yamatokoriyama, Nara 639-1080, Japan

Abstract:

A code review is one software review in which the developer closely reads the source code to find bugs in the source code. By performing the review in the early stages of development, it is possible to find as many bugs as possible at an early stage and to reduce the effort and cost of bug elimination. The purpose of this study is to find frequent patterns from eye movement data at code review to clarify the effect on the bug detection rate. In the experiment, we use the eye gaze data that recorded by an eye-measuring device during a task in which two source codes with multiple bugs, a design document, and a text file used by the program are presented (including the number of subjects and the number of bugs detected) during a code review when the subjects are divided into two groups: one group that receives instructions during the review and one group that does not receive instructions. We use cSPADE as a pattern mining technique to extract frequent patterns. The result of the analysis showed several frequent patterns related to the difference in the bug detection rate. The gaze pattern to alternately check the Caller-Callee relationship between source code and design documents was effective in detecting unnecessary variable declarations and errors in conditional statements. In addition, repeatedly moving back and forth between methods in a Caller-Callee relationship had a negative effect on the detection rate of incorrect method invocations.

Keywords: software review, code review, fault detection, eye movement analysis;

関連業績リスト

1. 二宮 凌真, “視線遷移データを用いたコードレビューの頻出パターン分析,” 第 3 ブロック専攻科研究フォーラム, 2022.

目次

図目次

表目次

1. 序論

ソフトウェアレビューはソフトウェアの開発においてプログラム中に混入しているバグを発見するために開発者がソースコードや設計書を精読する作業である．ソフトウェアレビューの 1 つに，コードレビューがあり，ソースコード中に混入しているバグを発見する目的で，開発者がソースコードを精読する作業である．この作業を開発の初期段階に実施することで，できるだけ多くのバグを早期に発見したり，バグの除去にかかる労力やコストを削減できる．

過去のコードレビューに関する研究として，應治は被験者を 1) 読み方を指示するグループと 2) 指示しないグループに分けたレビュー実験を実施し, レビュー効率とバグの指摘精度を比較するとともに，レビュー中の視線を計測した [2]．読み方を指示するグループには従来研究の結果から作業効率を向上させると考えられるレビューの具体的な手順を指示した. 各被験者のコードレビューにおける視線移動（どの文書のどの行を見たか）から多くの被験者に見られる共通した視線移動のパターン（頻出パターン）を複数抽出した．その結果，指示ありグループでは変数やメソッドが設計書の通りに使用されていることの確認や変数やプログラムの流れに沿って順に確認する視線移動が確認された．指示なしグループではプログラムの内容や設計書の構造を考慮せずに確認する視線移動が確認された．また，バグ検出率においては，経過時間の前半には指示なしグループ，後半には指示ありグループの方が高くなっていることが分かった．しかし，このような指示ありグループと指示なしグループのそれぞれに見られるパターン有無とレビュー効率や指摘精度の関係は明らかになっていない.

コードレビューの視線分布に関する研究として，伊藤らはプログラムの理解においてソースコードの構成要素の重要度を定量化し, 比較するためのフレームネットワークを作成し, レビュアーが注視する領域の視線移動の統計を取った [3]．その結果，レビュアーの注視点を表示するソースコード上に表示する注視度マップにおいて，特に注視度の高い場所が if や while などの直感的に重要度の評価が高いトークンと一致している．また，プ

ログラミング経験の少ない被験者の視線移動では目立った注視点がなく，ソースコードの領域全体を走査していることが分かった．

本研究では，應治の研究で分析されていなかった指示ありグループの頻出パターンと指示なしグループの頻出パターンの内容に注目して，それらの頻出パターンの中からバグ発見におけるレビュー効率の向上に効果的，または悪影響を及ぼすパターンについて分析し，明らかにすることを目的とする．

これらを明らかにすることで，コードレビューにおける最適なレビュー手法の設計に応用できる利点がある．本研究の成果が活用できる場面として，ソフトウェア開発組織におけるシステム開発が挙げられる．新入社員の研修において，第三者が書いたソースコードをレビューする時には，あらかじめ設定されたレビュー方法でレビューを行っている．しかし，そのレビュー方法でコード内に混入したすべてのバグを発見できるとは限らないことが問題である．バグ発見の効率化に有効なレビューパターンが定義できれば，レビュー効率やバグ検出率の向上につながると考えられる．

本研究では，指示ありグループと指示なしグループの時系列情報に変換した視線データにおいてそれぞれの頻出パターンを分析し，そのパターンから，それぞれのグループの視線データから該当する被験者を出力し，指示したことによって該当者率が増加したパターンや減少したパターンを分析する．提案手法では，視線移動の差にバグ発見に有効，無効なパターンが現れると考え，視線移動がレビュー対象となるドキュメント間やドキュメント内をどのような順で遷移するかを調べる．その視線移動の遷移を分析するための手法として時系列パターンマイニングを用いる．また，ソースコードのバグ検出率と頻出パターンの該当者率との関係を調べ，レビュー効率やバグ発見に効果的または悪影響なパターンを分析する．

以下，2 章では分析に関する準備について説明し，3 章では実験で使用するデータや分析手順について説明し，4 章に分析結果を述べた後，それに対する考察を行い，5 章では結論と今後の課題について述べる．

2. 準備

2.1 視線計測

視線計測 (アイトラッキング) [4] は人間の瞳の動きを追跡する技術のことで，頭部と眼球の運動を検知し，その人が何を見ているかをリアルタイムで追跡する生体計測手法である．主要な計測方法として赤外線カメラを用いて顔と瞳孔の位置を検出するアイトラッカーと呼ばれるデバイスを用いる．この装置で計測することで，視覚化されたヒートマップや視線の動きを追って表したゲイズプロット，瞬きの回数などの情報を得られ，人がどの部分にどのくらいの時間注目しているかや脳の動きと組み合わせることで興味や関心，記憶への定着度などが分かる．

2.2 時系列パターンマイニング

パターンマイニングとは, パターンと呼ばれる膨大な数の組合せ的規則の中から, 重要なものだけを効率的に取り出すことを目的とする技術の総称のことである [5]．その技術の 1 つに, 時系列パターンマイニング [6] があり, 時系列のデータベースから特定のしきい値を満たす出現順序を維持した部分文字列を時系列パターンとして発見する．時系列データベース (SDB) は複数の時系列データにより構成されていて，$SDB = < s_1, s_2, \cdots, s_n >$ と示される．s_k は時系列 ID を持ち, その ID 順で並んだ時系列データで構成されている．

時系列パターンマイニングの中で, 本研究で使用するパターン発見アルゴリズムは cSPADE [1] である．このアルゴリズムは, 入力として表 2.1 のような系列データベースと最小支持度を受け取る．系列データベースは, 複数の系列データから構成されており，各系列データは系列 ID(SID)，時間あるいは出現順序 (EID)，アイテムの集合 (Items) の 3 つで構成されている．系列データベースから表 2 のようにアイテムごとに SID と EID を作成したリストを作成し，そのリスト同士を結合していくことによって新しい系列パター

ンを検出する．例として，"A","B"というパターン長 1 の系列パターンから"A B"というパターン長 2 の系列パターンを得る際に作成されるリストを表 2.3 に示した．また，支持度とは系列パターンの出現頻度を表す値のことである．例として，表 2.1 の"D B A"というパターンは SID=1,4 で見られるため，この系列パターンの支持度は 0.5 である．リストの結合による新たな系列パターンを検出した際には，最小支持度未満の系列パターンはその時点で除外される．

表 2.1　系列データベース ([1] から引用)

SID	EID(時間)	Items
1	10	C D
	15	A B C
	20	A B F
	25	A C D F
2	15	A B F
	20	E
3	10	A B F
4	10	D G H
	20	B F
	25	A G H

表 2.2　表 2.1 から作成される SID と EID をまとめたリストの一部

A		B		
SID	EID(時間)	SID	EID(時間)	
1	15	1	15	...
	20		20	
	25	2	15	
2	15	3	10	
3	10	4	20	
4	25			

表 2.3　表 2.2 の A,B に関するリストを結合して得られるリスト

A B	
SID	EID
1	20

2.3　レビューにおける指示の有無

應治の研究 [2] でのコードレビューにおける指示ありグループと指示なしグループ [7] は, 指示ありグループと指示なしグループの全被験者に 2 つの Java のソースコードを対象としたレビューを行ってもらい，各被験者のレビュー効率を計測した. そのデータを基にして，2 つのグループ間で被験者の能力差が現れないようにするために，バグ検出率が等しくなるようグループを分けた．分割したグループから指示ありグループにレビューに関する指示を伝え，指示なしグループには何も伝えず，それぞれのグループで Java のソースコードのレビューを行った.

コードレビューにおける指示はバグの検出において，プログラムの制御フローやソースコード上のクラス・メソッドの記述位置という構造が設計書と異なっていないかどうかの確認を促すために行われている．バグの検出には，設計書を含むコードレビューにおいて設計書に記述されたプログラムの機能や構造を理解する必要がある．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．また，設計書を読むことでプログラム自体の理解が効率化することに加え，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．コードレビューの従来の指示内容は，「設計された内容は完全にコーディングされているか」と言ったコードの完全性に関する項目や，「＝，＝＝，｜｜などの演算子は正しく使われているか」と言ったプログラミング言語に依存する項目について，具体的にどう読めば，確認できるか書かれておらず，読み方がレビュアーによって変わり，レビュー効率にも差が出る.

上記のことから，指示ありグループの作業者にコードレビュー開始時に，ソースコードの構造理解や設計書の内容把握によるプログラムへの理解度，レビュー効率が向上するという仮説と設計書とソースコードの整合性を確認するよう読み方を基にして設定した指示の具体的な内容を伝えた.

3. 分析

3.1 分析対象

3.1.1 視線データ

本研究では, 過去の研究 [2] で計測された視線データを使用する．指示ありグループの被験者に伝えられたコードレビューに関する具体的な指示は 2.3 節で述べた指示によるレビュー効率の向上させる仮説を基にして設定したものである．この計測における被験者の人数は指示ありグループ 6 人，指示なしグループ 8 人，合計 14 人である. この視線データは，ファイルに記録された賃貸物件から条件にあう物件を検索し，表示するという java プログラムや設計書，物件ファイル，チェックリストを対象とするソースコードレビューをした時のものである. 各ファイルの詳細については 3.1.2 節で説明する.

このデータの各行にはある一定時間以上注視した部分の行番号とその番号に対応するファイル名などが記録されている．そのデータの中から，各ファイルに記載されたメソッドやメソッドに関する説明を 1 つのブロックとして分割し，各ブロックにブロック記号を割り当てたものに対して視線の時系列情報記号に変換した視線データを使用する.

2.3 節で述べた指示ありグループに伝えたコードレビューに関する具体的な指示は図 3.1 に示す 3 点であり，コードレビュー開始前に実験者が口頭で伝える.

(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.

図 3.1　指示の内容

(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，(2) と (3) はソースコードの構造を理解してもらい，ソースコードと設計書の整合性を確認するための指示である．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一方で，指示することで手順を実行している間バグを指摘する作業を行わなくなり，指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．

3.1.2 レビュー対象プログラム

3.1.1 節におけるソースコード，設計書，データファイル，コードチェックリストの内容とソースコードのバグの内容をそれぞれ表 3.1，表 3.2 に示す．

表 3.1 各ファイルの内容

ブロック記号	ファイル名	行数	メソッド数	備考
D_m	Main.java	104	3	物件のデータ読み込み，検索処理などを行うソースコード
D_h	House.java	31	4	物件の駅からの距離，家賃，坪数の取得と物件名，坪数，駅からの距離，家賃を一行で表示するソースコード
D_s	設計書.txt	41	-	Main.java や House.java のメソッドごとの説明をまとめたもの
D_{bu}	bukken.dat	20	-	物件名, 坪数, 距離, 家賃のデータを記録されているもの
D_c	コードチェックリスト.txt	37	-	ソースコードにおける完全性, 初期化, メソッド呼び出し, 演算, データ・ファイル, 制御のチェック点が説明されているもの

表 3.2　ソースコードに混入するバグの内容

バグ	ソースコード	行番号	内容	備考
B_1	House.java	33	家賃 rent が表示されない	必要な変数の抜け
B_2	Main.java	11	物件の上限値 (MAXHOUSE) が違う	Main.java に対応する設計書の説明に書かれている数値と異なる
B_3		47	lower → upper	使用する変数が正しい使い方ではない
B_4		49	upper → lower	使用する変数が正しい使い方ではない
B_5		63	|| → &&	異なる演算子が使われている
B_6		84	不要な宣言	必要でない変数が使われていて，プログラムの動作に影響がない
B_7		91	&& $i > MAXHOUSE$ がない	While 文の条件が間違っている

また, 設計書，House,java，Main.java をブロックごとに分割した時の図をそれぞれ図 3.2, 図 3.3, 図 3.4 に示す．設計書は Main.java に関する説明はメソッドごとに分割し，House.java に関する説明はすべてのメソッドを 1 つのブロックにまとめた. House.java はクラス変数の定義，コンストラクタの生成，メソッドの 3 つのブロックに分割した.Main.java は空白行で区切ってメソッドやコードを分割した. また, 各ブロックの行数や内容についてまとめた表を表 3.3 に示す．また，表 3.3 をもとにブロック記号に変換した視線データに対して，a) メソッドの呼び出し関係にあるブロックを移動する視線 (Caller-Callee) [8], b) コード片と対応する設計書の段落を移動する視線 (Code-Design), c) 単一のドキュメント内で上下に位置するブロックを移動する視線 (Up-Down)，d) その他の 4 項目に a〜c の方向を加味した計 7 種類の視線移動の時系列情報に変換するための表を表 3.4 に示す.

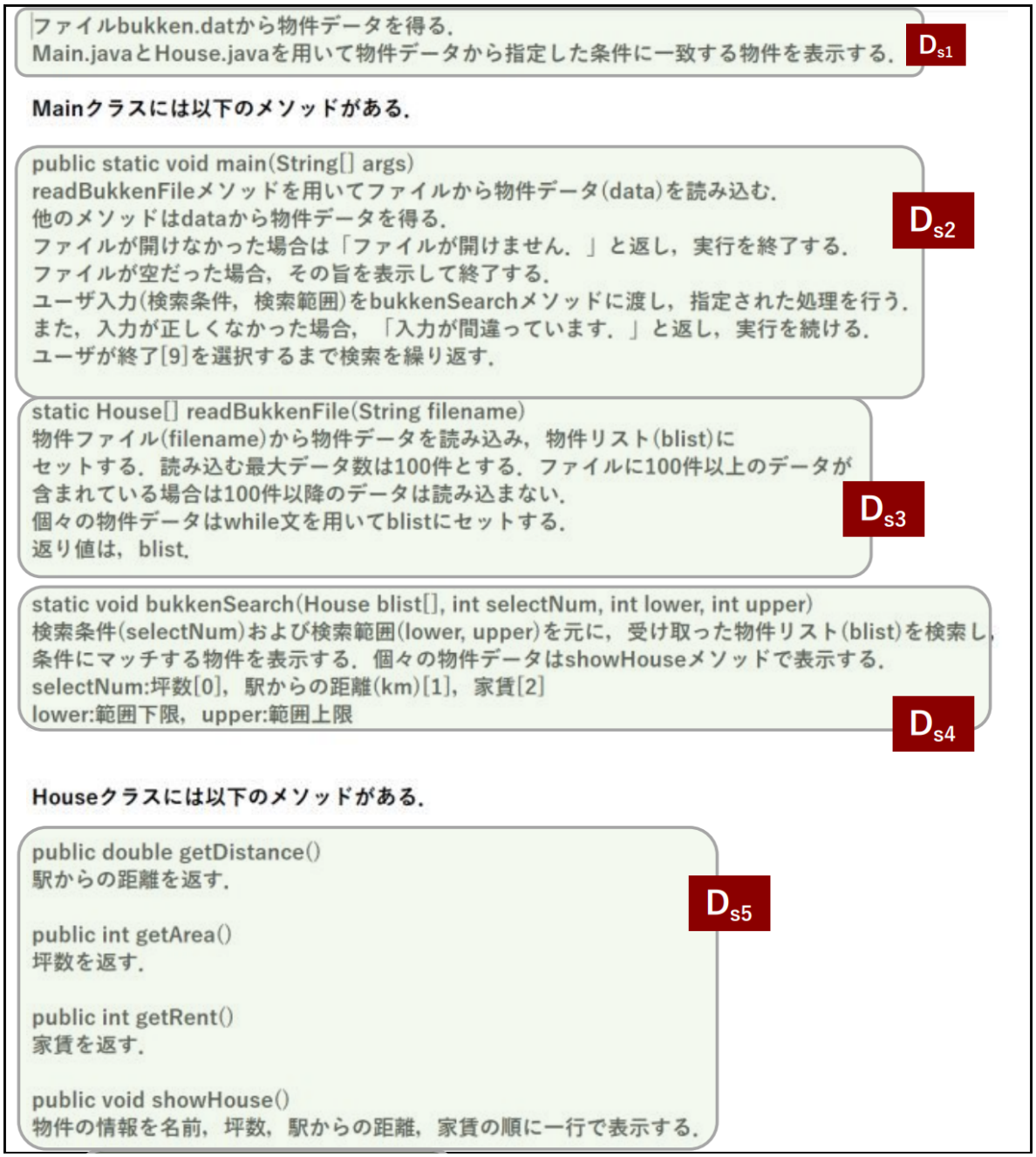

図 3.2　設計書のブロック分割図

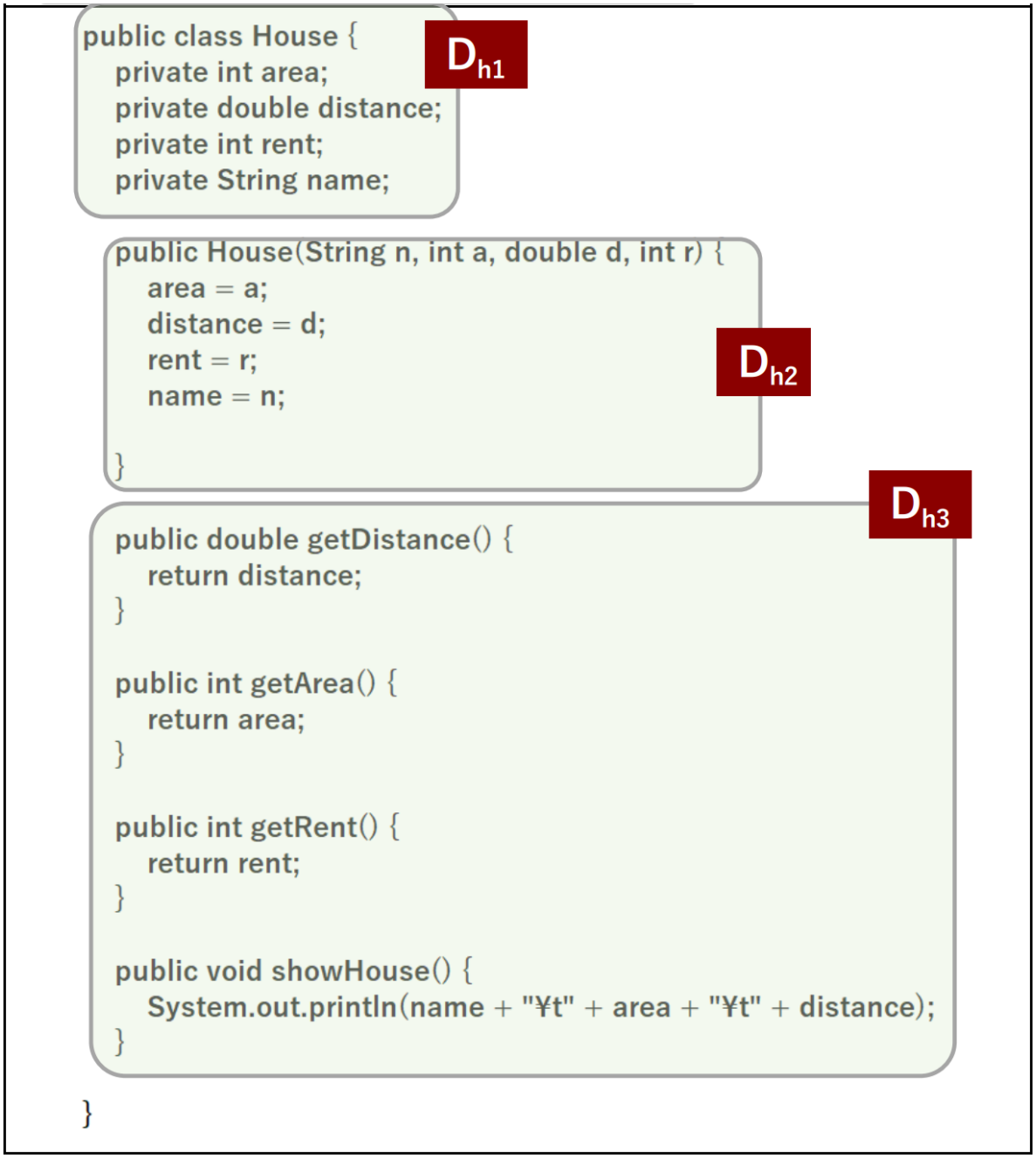

図 3.3　House.java のブロック分割図

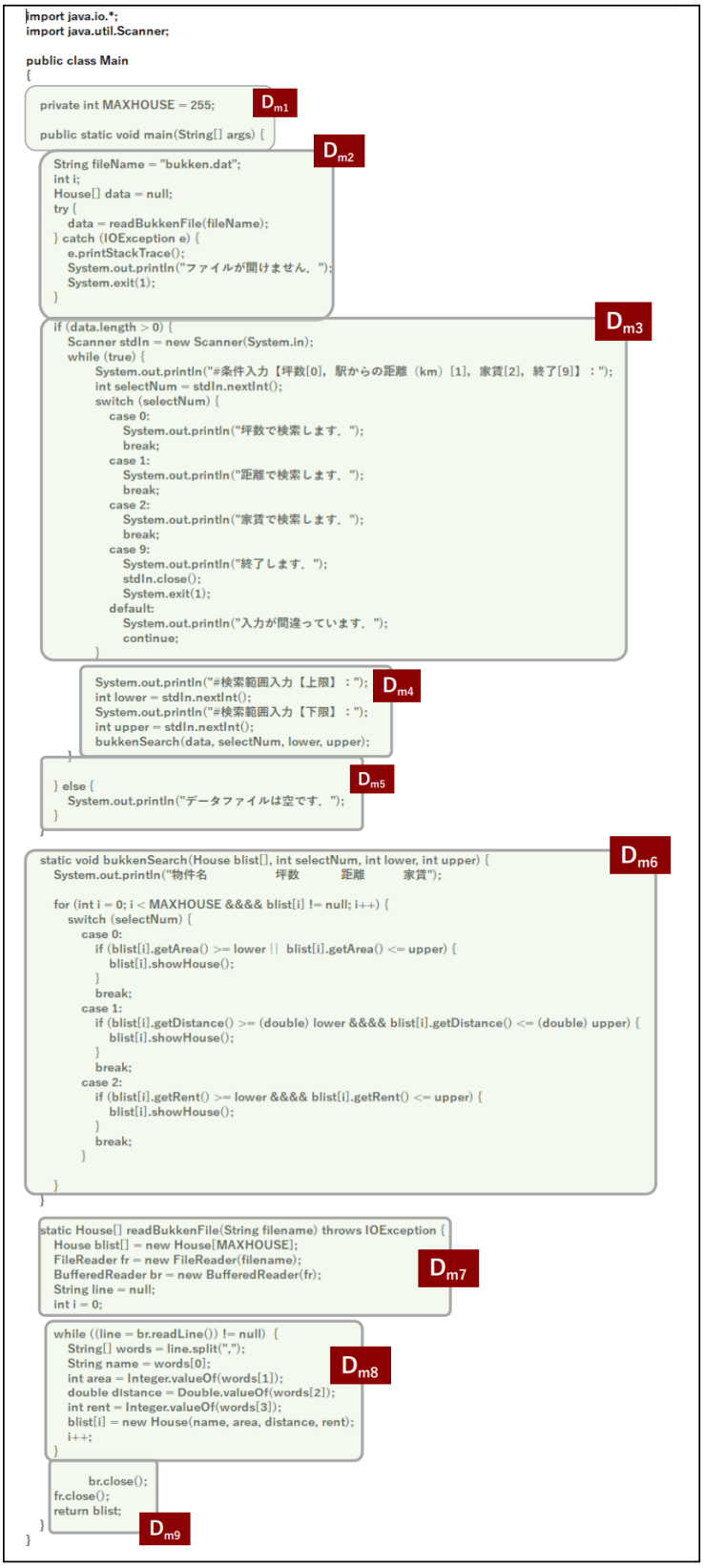

図 3.4　Main.java のブロック分割図

表 3.3　ファイルブロック分割の内容

ブロック記号	ファイル名	行範囲	内容
D_0	java ファイル		例外の行 (ブロック番号 1〜19 のうちどれにも属さないもの)
D_{s1}	設計書.txt	ℓ 1 ∼ 2	プログラムの簡単な処理の流れの説明
D_{s2}		ℓ 6 ∼ 13	public static void main(String[] args) の説明
D_{s3}		ℓ 15 ∼ 20	static House[] readBukkenFile(String filename) の説明
D_{s4}		ℓ 22 ∼ 26	static void bukkenSearch(House blist[], int selectNum, int lower, int upper) の説明
D_{s5}		ℓ 31 ∼ 41	House クラスで使用するメソッドの説明
D_{h1}	House.java	ℓ 2 ∼ 5	クラス内変数の宣言
D_{h2}		ℓ 7 ∼ 13	インスタンス生成
D_{h3}		ℓ 15 ∼ 29	駅からの距離, 坪数, 家賃の返却メソッド 物件の情報を名前, 坪数, 駅からの距離, 家賃の順に一行で表示メソッド
D_{m1}	Main.java	ℓ 6 ∼ 9	最大物件数の定義
D_{m2}		ℓ 10 ∼ 20	入力ファイルや変数, 配列の定義, ファイルの存在確認処理
D_{m3}		ℓ 22 ∼ 44	物件情報があるときの検索分岐処理
D_{m4}		ℓ 46 ∼ 51	検索数の上限, 下限を決定と bukkenSearch メソッドへ受け渡し処理
D_{m5}		ℓ 53 ∼ 56	物件情報がない旨の表示
D_{m6}		ℓ 58 ∼ 81	static void bukkenSearch(House blist[], int selectNum, int lower, int upper)
D_{m7}		ℓ 83 ∼ 88	static House[] readBukkenFile(String filename) throws IOException (変数定義)
D_{m8}		ℓ 90 ∼ 98	static House[] readBukkenFile(String filename) throws IOException (検索処理)
D_{m9}		ℓ 100 ∼ 103	ファイルクローズ, 条件と一致した物件のリスト返却
D_{bu1}	Bukken.dat	ℓ 1 ∼ 20	物件名, 坪数, 距離, 家賃のデータをまとめたもの
D_{c1}	コードチェックリスト.txt	ℓ 1 ∼ 37	完全性, 初期化, メソッド呼び出し, 演算, データ・ファイル, 制御のチェック点をまとめたもの

表 3.4　視線の時系列情報記号への変換表

時系列情報記号	該当のブロック遷移
RE,ER	$D_{s1} \Leftrightarrow D_{s2}, D_{s1} \Leftrightarrow D_{s3}, D_{s1} \Leftrightarrow D_{s4}, D_{s1} \Leftrightarrow D_{s5},$ $D_{s2} \Leftrightarrow D_{s3}, D_{s2} \Leftrightarrow D_{s4}, D_{s4} \Leftrightarrow D_{s5},$ $D_{m2} \Leftrightarrow D_{m7}, D_{m2} \Leftrightarrow D_{m8}, D_{m2} \Leftrightarrow D_{m9},$ $D_{m4} \Leftrightarrow D_{m6}, D_{m4} \Leftrightarrow D_{h2}, D_{m8} \Leftrightarrow D_{h3}$
DC,CD	$D_{s1} \Leftrightarrow D_{m1}, D_{s1} \Leftrightarrow D_{m2}, D_{s1} \Leftrightarrow D_{m3}, D_{s1} \Leftrightarrow D_{m4},$ $D_{s1} \Leftrightarrow D_{m5}, D_{s1} \Leftrightarrow D_{m6}, D_{s1} \Leftrightarrow D_{m7}, D_{s1} \Leftrightarrow D_{m8},$ $D_{s1} \Leftrightarrow D_{m9}, D_{s1} \Leftrightarrow D_{h1}, D_{s1} \Leftrightarrow D_{h2}, D_{s1} \Leftrightarrow D_{h3},$ $D_{s2} \Leftrightarrow D_{m1}, D_{s2} \Leftrightarrow D_{m2}, D_{s2} \Leftrightarrow D_{m3}, D_{s2} \Leftrightarrow D_{m4},$ $D_{s2} \Leftrightarrow D_{m5}, D_{s2} \Leftrightarrow D_{m6}, D_{s2} \Leftrightarrow D_{m7}, D_{s2} \Leftrightarrow D_{m8},$ $D_{s2} \Leftrightarrow D_{m9}, D_{s3} \Leftrightarrow D_{m2}, D_{s3} \Leftrightarrow D_{m7}, D_{s3} \Leftrightarrow D_{m8},$ $D_{s3} \Leftrightarrow D_{m9}, D_{s4} \Leftrightarrow D_{m6}, D_{s5} \Leftrightarrow D_{h1}, D_{s5} \Leftrightarrow D_{h2}, D_{s5} \Leftrightarrow D_{h3}$
UP,DW	$D_{s2} \Leftrightarrow D_{s1}, D_{s3} \Leftrightarrow D_{s2}, D_{s4} \Leftrightarrow D_{s3}, D_{s5} \Leftrightarrow D_{s4},$ $D_{m2} \Leftrightarrow D_{m1}, D_{m3} \Leftrightarrow D_{m2}, D_{m4} \Leftrightarrow D_{m3},$ $D_{m5} \Leftrightarrow D_{m4}, D_{m8} \Leftrightarrow D_{m7}, D_{m9} \Leftrightarrow D_{m8}$
OT	上記以外

3.2　分析手順

分析の流れを表した図を図 3.5 に示す. まず, 表 3.4 をもとに指示ありグループと指示なしグループの視線データを変換し，それぞれの視線データから頻出パターンを抽出するために 2.2 節の cSPADE を用いて，グループの人数の内，半数以上が該当する系列パターンを出力する．また，抽出するパターンの最長パターン長 max を 10，最小パターン長 min を 1 とした. それぞれのグループで抽出された頻出パターンの内，"OT"を 1 つ以上含むものと図 3.6 に示すように他の頻出パターンに内包されるパターン長が短いパターンは分析の対象から除外する．これらのパターンを除外する理由として，"OT"を含むパターンは瞬きなどの計測上の誤差や次に読むブロックへの移動といったノイズであると考え，取り除く必要があると考えたからである．また，内包されるパターン長の短いパターンは同様の意味を示すパターンを統合することで分析数を減らし，より複雑な意味を持つパターン

だけを中心に分析することができるからである．これらの処理で残った頻出パターンについてそれぞれのグループ内で類似するパターン列をグループ化して分析を行う．

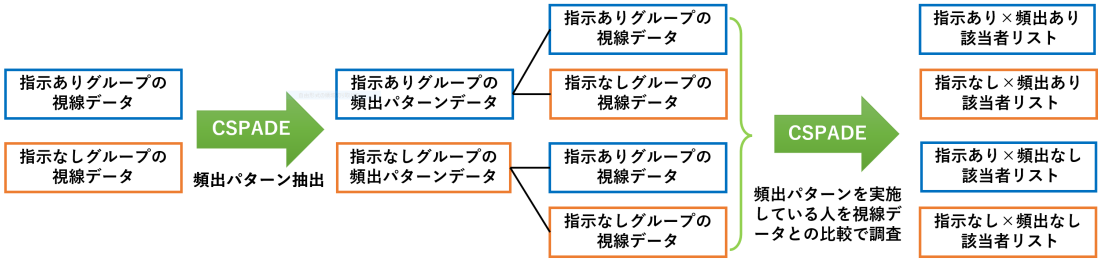

図 3.5　分析の流れ

パターン列 $A, B(Len(A) < Len(B))$ が以下を満たすとき A を頻出パターンから除外する (k 番目の要素を A_k, B_k とする)

(1) **後ろに長いパターンに内包する場合**

$\bigcap_{k=1}^{Len(A)} (A_k = B_k) = TRUE$ で,$A_{Len(A)+1}$ が存在しない, かつ $B_{Len(A)+1}$ 以降が存在する場合

(2) **前に長いパターンに内包する場合**

$\bigcap_{k=0}^{Len(A)-1} (A_{Len(A)-k} = B_{Len(B)-k}) = TRUE$ で,A_0 が存在しない, かつ $B_{Len(B)-Len(A)}$ 以前が存在する場合

図 3.6　内包できるパターン長の短い頻出パターンを除外する方法

次に，コードレビューにおいて指示の有無によって実行する人数が増加したパターンと減少したパターンを分析するために，cSPADE を用いて，各頻出パターンが各被験者の視線データの中に含まれているかどうかを出力する．

この作業を，指示によって，指示ありグループの頻出パターンと指示なしグループの頻出パターンにおける各パターンがそれぞれのグループだけに現れるパターンかどうかを調べる．図 3.7 に示す 4 つの場合に基づいて，指示ありグループと指示なしグループの各頻出パターンについて，それぞれのグループ内の該当者率 ($\frac{\text{頻出パターンを実施した人数}}{\text{グループ内の被験者の人数}}$) と各頻出パターンの該当者率の差 (指示ありグループ − 指示なしグループ) を算出し，分析する．

(1) 指示あり頻出パターン×指示あり視線データ
(2) 指示あり頻出パターン×指示なし視線データ
(3) 指示なし頻出パターン×指示あり視線データ
(4) 指示なし頻出パターン×指示なし視線データ

図 3.7　パターン該当者を探索する場合分け

最後に，各頻出パターンの実施の有無と各バグの検出の有無を集計する．また，バグ検出率の差 (頻出パターンの該当者の内，バグを発見した人数の割合と該当者の内，バグを発見しなかった人数の割合の差) を算出し，パターンを実施した被験者のバグ検出率が +50% 以上および，-50% 以下を含むパターンをそれぞれ効果的な頻出パターン，悪影響な頻出パターンとして調べる．

4. 結果と考察

4.1 指示ありと指示なしグループの頻出パターンの比較

指示ありグループと指示なしグループの各被験者の視線データに対して cSPADE による頻出パターンの抽出を行い，その中から, 内包できるパターン列を除外した結果，指示ありグループの頻出パターンの総数は 75 個，指示なしグループの頻出パターンの総数は 83 個になった．それぞれのグループ内で類似する頻出パターンをグループ化した結果を表 4.1 に示す.

表 4.1　指示ありグループと指示なしグループの頻出パターンのグループ化

	グループ	パターン数	内容
指示あり	I_1	6	RE,ER が連続するパターン列
	I_2	11	RE,ER と CD,DC のみを含むパターン列
	I_3	7	RE,ER と CD,DC の後ろに UP,DW が連続するパターン列
	I_4	5	RE や ER の後ろに,UP,DW が連続するパターン列
	I_5	5	CD や DC の後に,UP,DW が連続するパターン列
	I_6	11	UP,DW が連続した後に,CD や DC を含んでいるパターン列
	I_7	19	UP や DW から始まるパターン列
	I_8	6	ER から始まるパターン列
	I_9	5	ER から始まり,UP,DW や DC,CD で終わるパターン列
指示なし	NI_1	4	RE,ER が連続するパターン列
	NI_2	23	UP,DW が連続するパターン列
	NI_3	6	UP や DW から始まり, その後に RE,ER が連続するパターン列
	NI_4	5	UP や DW から始まり,2 番目に CD を含むパターン列
	NI_5	11	UP,DW が連続した後に,CD,RE,ER で終わるパターン列
	NI_6	6	RE や ER で始まり, その後に UP,DW が連続するパターン列
	NI_7	9	連続する RE や ER の間に,UP,DW が含まれるパターン列
	NI_8	6	RE や ER が続いた後に,DC が含まれるパターン列
	NI_9	7	RE,ER と CD,DC のみを含むパターン列
	NI_{10}	6	CD や DC から始まり, その後に UP,DW が連続するパターン列

この表から，指示ありグループと指示なしグループそれぞれの頻出パターンに共通する点として，I_1 や NI_1 というように RE と ER が連続しているパターン列が含んでいることであり，指示ありグループの方がパターン数として多くなっていることがいえる．I_2 や NI_9 というように RE,ER と CD,DC のみが含まれているパターン列があり，ブロック間での遷移関係にある視線移動をしていることがいえる．I_5 や NI_{10} というように CD や DC から始まり，その後に UP,DW が連続しているパターン列があり，遷移した先でブロックの周辺を読み進める視線移動をしていることがいえる．

指示ありグループと指示なしグループそれぞれの頻出パターンで異なる点として，指示なしグループの NI_2 のような UP や DW が連続するパターンが指示ありグループには含まれていないことである．

これらのことから，指示ありグループと指示なしグループの頻出パターンにおいて，指示によって，ドキュメント内をひたすら上下に視線移動させることが減り，遷移関係のあるブロックの内容をより注意深く読み進めていることがいえる．

表 4.1 の中から，I_1 と NI_1，I_2 と NI_9，I_5 と NI_{10} に含まれている頻出パターンの一部をそれぞれ図 4.1，図 4.2, 図 4.3, 図 4.4，図 4.5, 図 4.6 に示す．

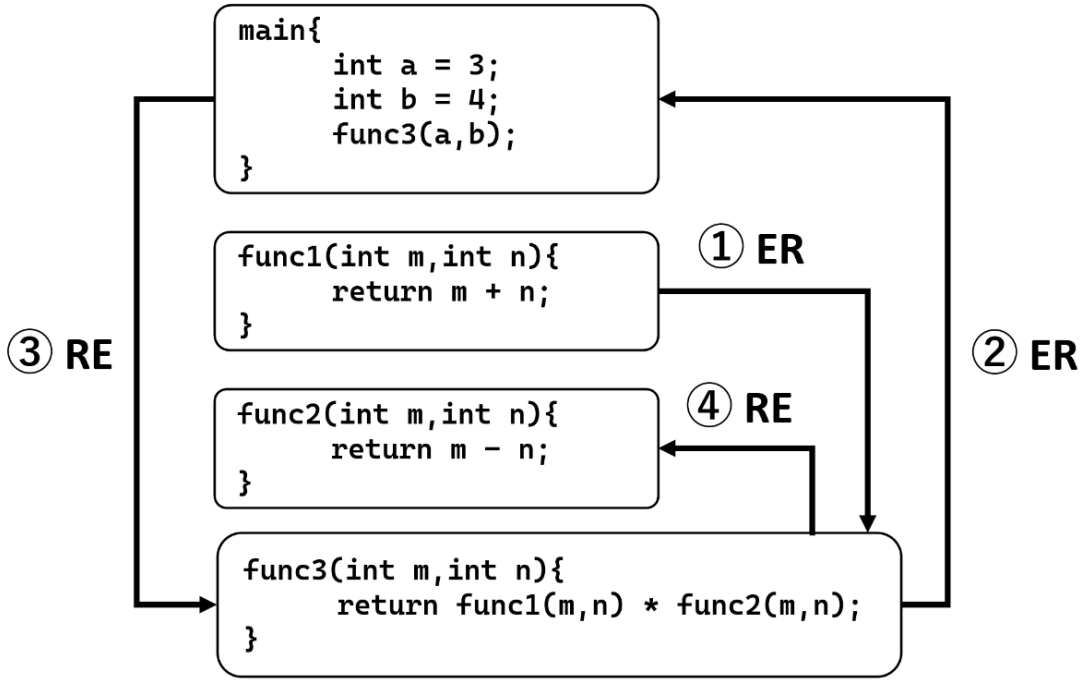

図 4.1　I_1 に含まれている頻出パターン

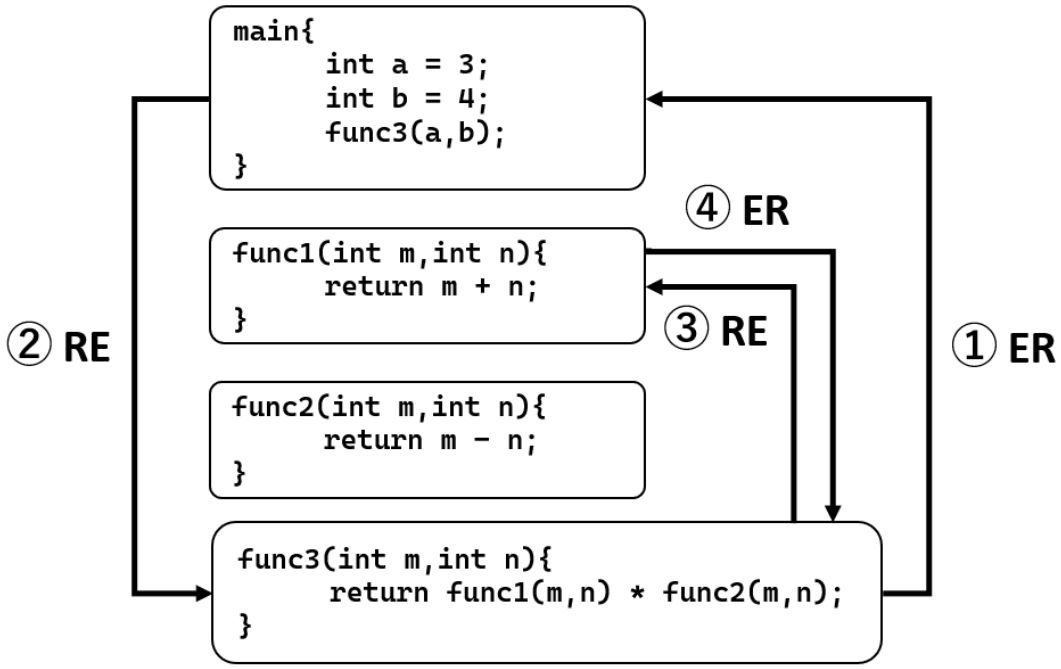

図 4.2　NI_1 に含まれている頻出パターン

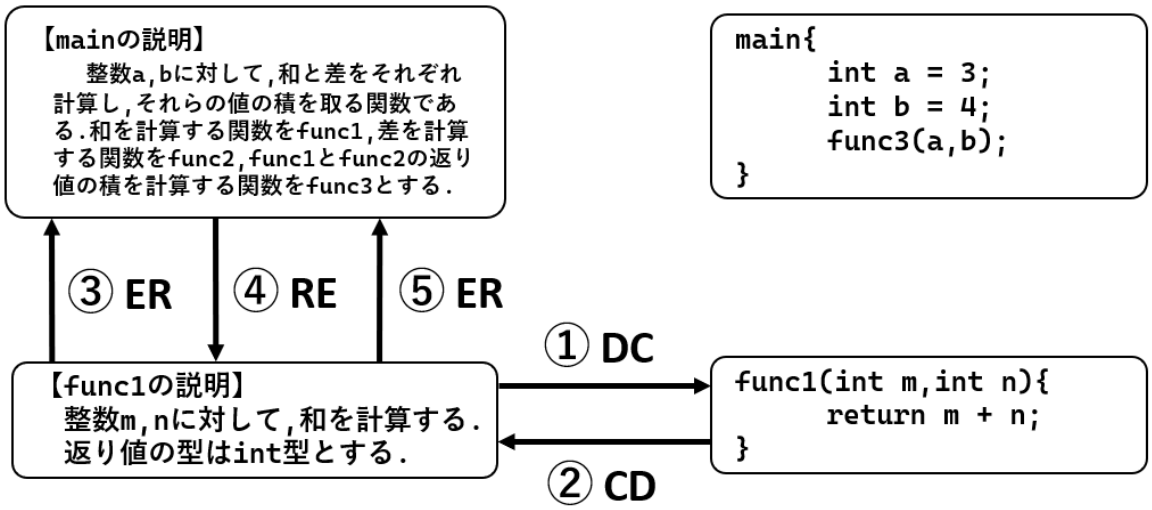

図 4.3　I_2 に含まれている頻出パターン

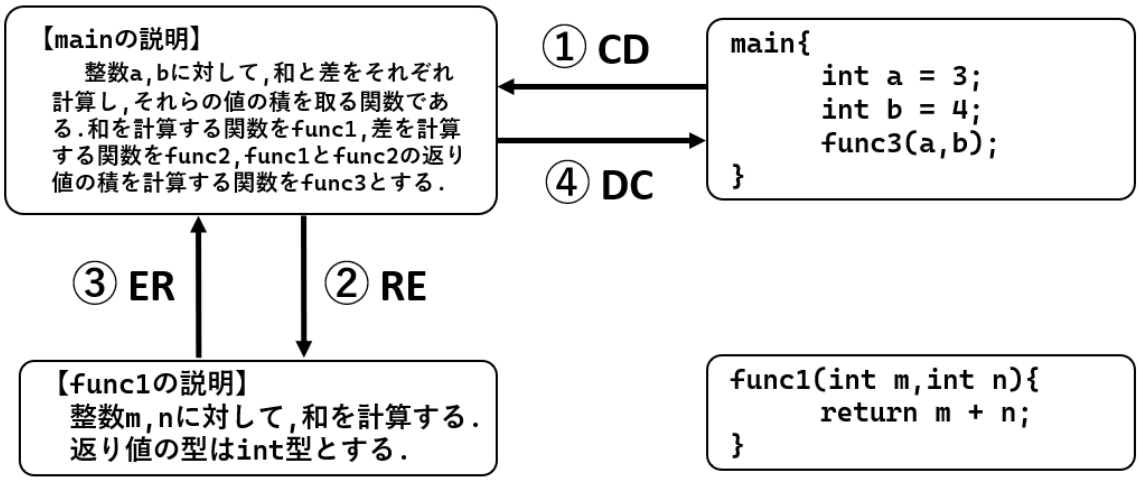

図 4.4　NI_9 に含まれている頻出パターン

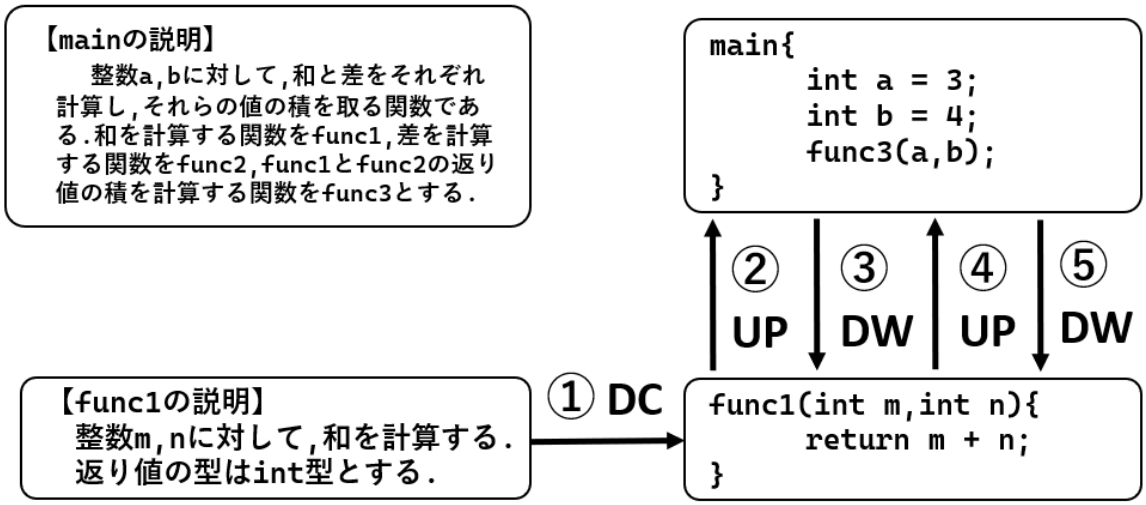

図 4.5　I_5 に含まれている頻出パターン

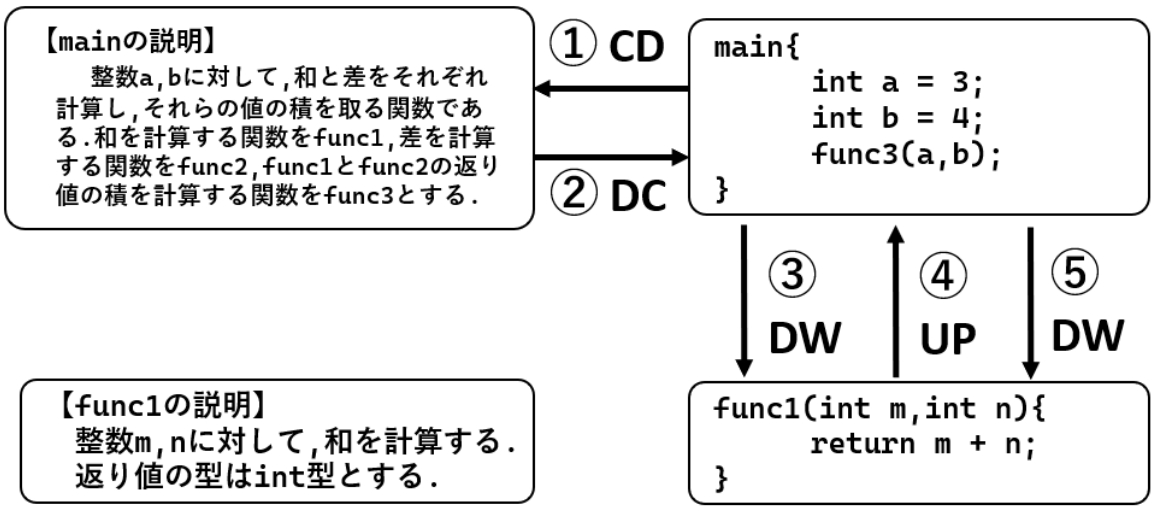

図 4.6　NI_{10} に含まれている頻出パターン

各パターンを表す図の矢印はブロック間を移動する視線移動の順序とブロック間の関係を表している．また，丸四角はパターンを示すための簡易的な設計書やソースコードの要約を表し，実際のコードや設計書の内容とは異なる．

図 4.1 や図 4.2 のパターンは，Caller-Callee 関係にあるメソッドのコード (main,func1, func2,func3) を順に行き来する視線であり，メソッド同士の処理の関わりを理解するための視線移動と考えられる．この視線は設計書の中でも同様に出現すると考えられる．

図 4.3 や図 4.4 のパターンは，Code-Design 関係にあるメソッドの設計と対応するコード (func1) を行き来した後に，その設計と Caller-Callee 関係にあるメソッド (main) の設計を繰り返し行き来する視線であり，ある 1 つのメソッドの設計について理解した後に，Caller-Callee 関係を持つ main について読み進める視線移動であると考えられる．また，メソッドのソースコード (main) と対応する設計を順に見た後，その設計と Caller-Callee 関係にあるメソッド (func1) の設計を行き来し，もとのメソッドのコード (main) に戻る視線であり，メソッドのコードからこのプログラムがどのような設計であるかを遷移関係に基づいてたどっていきながら理解する視線移動をしていると考えられる．

図 4.5 や図 4.6 のパターンは，Code-Design 関係にあるメソッド (func1) の設計，対応するソースコードを順に見た後，周辺のメソッド (main) を繰り返し行き来する視線であり，設計の内容が簡単なメソッド (func1) を中心に周辺のメソッドを注意深く確認するように読み進める視線と考えられる．また，Code-Design 関係にあるメソッドのコードと

対応する設計 (main) を行き来した後，周辺のメソッド (func1) を繰り返し行き来する視線であり，最初に呼び出されるメソッド (main) を中心に周辺のメソッドを確認するように読み進める視線移動と考えられる．

次に，指示ありと指示なしグループの頻出パターンにおいて, 指示ありグループの該当者率と指示なしグループの該当者率との差が多くなったパターンを表 4.2 に示す．2 列目のパターンはパターン列を示し，左から順に該当する視線移動があることを示す．該当者率の差が正の値のときには指示ありグループ，負の値のときには指示なしグループでパターンを実施する人数が多いことを表す．また，表の I_3 と NI_5 の頻出パターンについて図 4.7, 図 4.8 に示す．

表 4.2: 指示ありと指示なしグループの頻出パターンにおける該当者数と該当者率

グループ	パターン	該当者数		該当者率		
		あり	なし	あり	なし	差
I_1	RE,ER,RE,ER,RE,ER,RE,ER,RE	4	2	67%	25%	42%
I_2	DC,CD,ER,RE,ER	3	0	50%	0%	50%
	CD,RE,ER,RE,DC	3	0	50%	0%	50%
	CD,DC,CD,ER,RE	3	0	50%	0%	50%
I_3	RE,DC,UP,DW,DW,UP	3	0	50%	0%	50%
	RE,DC,DW,DW	3	0	50%	0%	50%
	CD,ER,RE,ER,DC,DW,UP,DW,UP	3	0	50%	0%	50%
	CD,DC,DW	4	5	67%	63%	50%
I_6	UP,DW,UP,DW,UP,DW,UP,DW,UP,CD	3	0	50%	0%	50%
I_7	UP,DW,UP,CD,RE	3	0	50%	0%	50%
	UP,DW,UP,CD,ER,RE	3	0	50%	0%	50%
	UP,CD,ER,RE,ER	3	0	50%	0%	50%
	DW,CD,DW	3	0	50%	0%	50%
	DW,UP,DW,UP,DW,RE	3	0	50%	0%	50%
	DW,UP,DW,UP,DW,UP,DW,UP,CD,RE	3	0	50%	0%	50%
NI_2	UP,UP,DW,UP,UP	1	5	17%	63%	-46%
	UP,UP,UP,DW,DW,UP,DW	0	4	0%	50%	-50%
	UP,UP,DW,UP,DW,UP,UP	0	4	0%	50%	-50%
	UP,DW,UP,UP,DW,DW,DW,DW	0	4	0%	50%	-50%
	DW,UP,UP,DW,UP	6	4	100%	50%	50%
NI_5	DW,DW,DW,RE	0	5	0%	63%	-63%
	UP,DW,DW,RE	0	4	0%	50%	-50%
NI_6	ER,UP,UP,UP,DW,DW	0	4	0%	50%	-50%
NI_7	ER,RE,ER,RE,DW,UP,ER	0	4	0%	50%	-50%

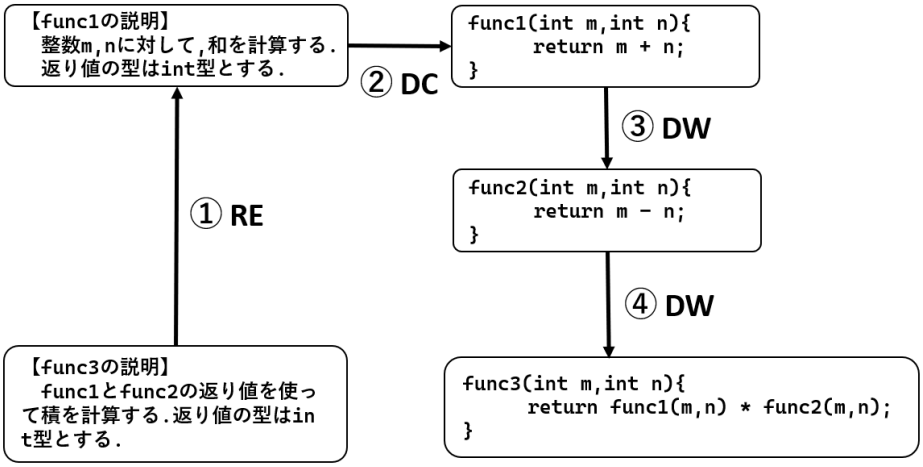

図 4.7　I_3 の頻出パターンの中で該当者率の差が大きいパターン

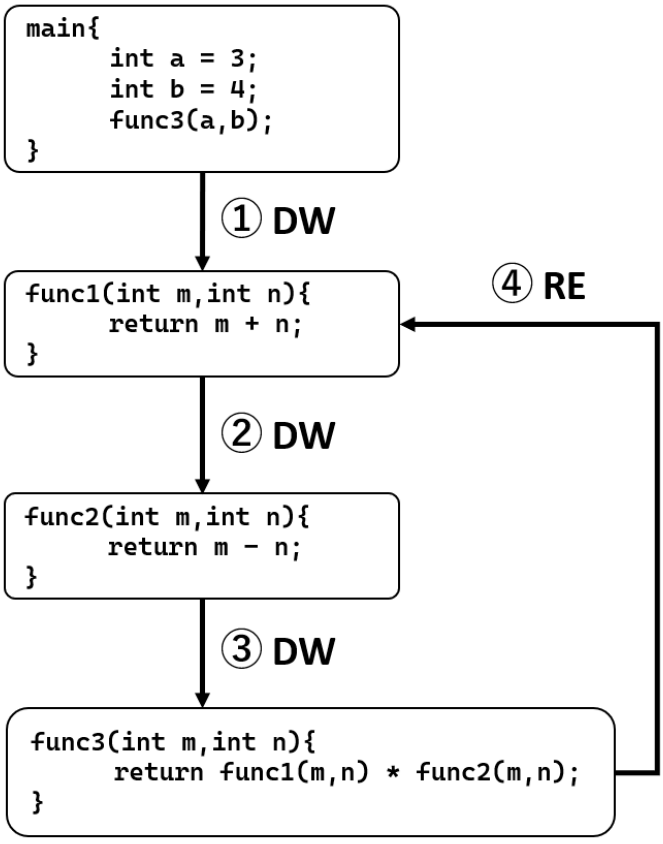

図 4.8　NI_5 の頻出パターンの中で該当者率の差が大きいパターン

表 4.2 の結果より，指示ありグループの該当者率が高くなったパターンは，RE や ER が連続するパターン列や Caller-Callee 関係や Code-Design 関係を持つブロックを移動する視線の前後に，上下に移動する視線を行っていることがいえる．これは，図 3.1 にあった指示からメソッドや設計書の内容を注意深く見ていると考えられる．指示なしグループの該当者率が高くなったパターンは, 上下に連続して移動する視線や Caller-Callee 関係を持つブロックを移動する視線の途中で上下に移動する視線があることがいえる．これは，レビューに対する具体的な指示がなく，自分なりのやり方でバグを探していることによって，1 つのブロックの内容を理解するのに時間がかかり視線を何回も上下に移動させていると考えられる．

図 4.7 のパターンは，Caller-Callee 関係にあるメソッド (func1,func3) の設計を順に見た後，対応するソースコード (func1) とその下に続くメソッド (func2,func3) を順に移動する視線である．これは，ある関数の内容を理解するにあたって，関係のあるものをすべてたどって読み進めていく視線移動を行うことによって，プログラムの全体を理解していると考えられる．

図 4.8 のパターンは，メインとなるコード (main) を始点として上から順に見た後，Caller-Callee 関係にあるメソッド (func1,func3) を見る視線である．これは，ソースコードがどういう構造なのかを確認していきながら，気になった行の内容について戻って内容を理解すること繰り返しながらプログラムの全体を理解する視線移動と考えられる．

4.2 バグ検出率と頻出パターン

指示ありグループと指示なしグループの頻出パターンにおいて各パターンのバグ検出率を調べた結果，指示ありの頻出パターンでは B_6 と B_7 の検出率が高く，B_1 と B_2 の検出率が低いことが分かった．また，指示なしの頻出パターンでは B_1 と B_6 の検出率が高いことが分かった．

指示ありグループと指示なしグループそれぞれの頻出パターンの該当者とバグ検出率との関係を調べ，その中でバグの検出率の差が +50% 以上および，-50% 以下となった一部のパターンをそれぞれ，図 4.9，図 4.10 に示す. また，それぞれのパターンにおけるの各バグの検出率を表 4.3 に示す.

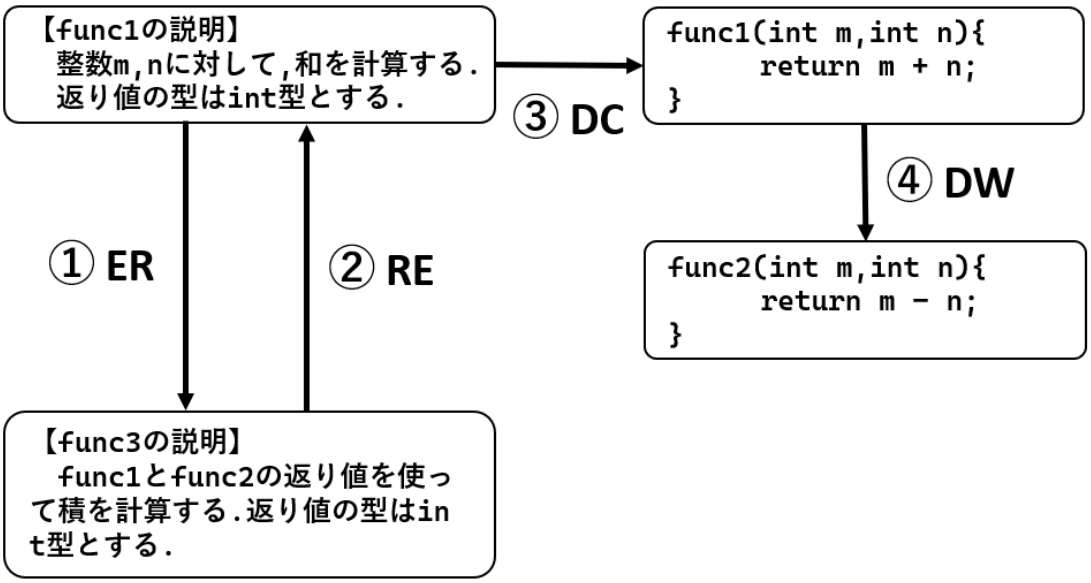

図 4.9　バグ検出率の差が +50% 以上の I_8 の頻出パターン

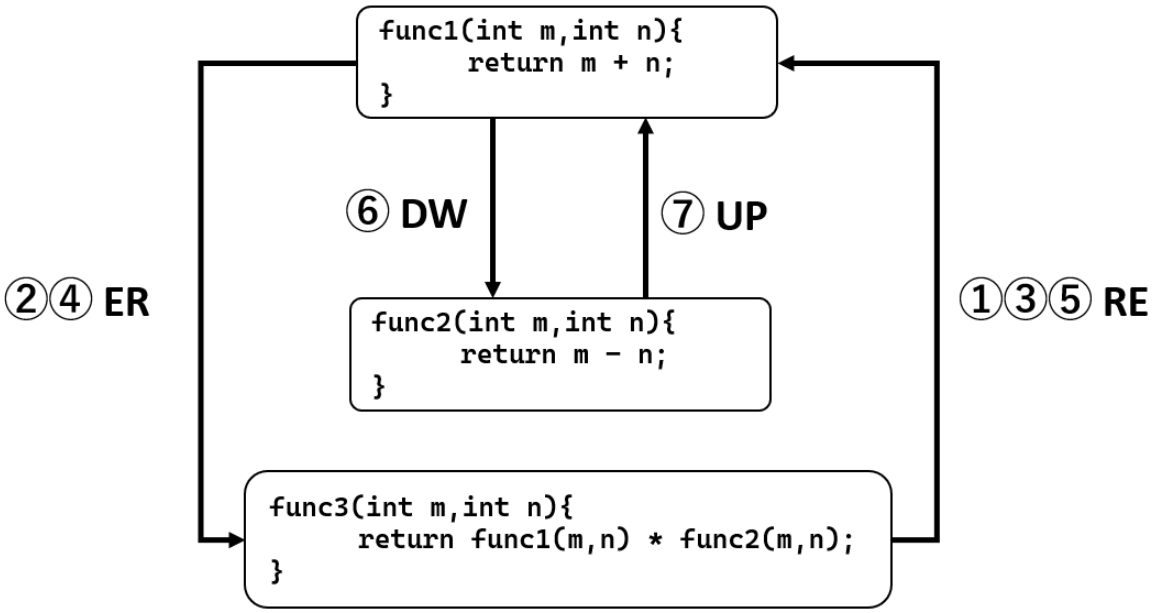

図 4.10　バグ検出率の差が −50% 以下の NI_7 の頻出パターン

表 4.3　図 4.8 と図 4.9 のパターンにおける各バグの検出率の差

バグ	パターン	
	図 4.9	図 4.10
B_1	-62%	54%
B_2	-67%	-58%
B_3	50%	-58%
B_4	50%	-58%
B_5	-8%	58%
B_6	67%	-58%
B_7	73%	21%

図 4.9 のパターンは，Caller-Callee 関係にあるメソッド (func1,func3) の設計を順に見た後に，対応するソースコード (func1) と同じメソッド (func3) から呼び出されるもう 1 つのメソッド (func2) を順に移動している視線である．このパターンは，B_6 と B_7 の検出率を増加させているのに対して，B_1 と B_2 の検出率を減少させているといえる．このことから，指示した状況においてこのパターンを実施することで，不要な変数の宣言や条件文の記述の間違いの検出に効果的であるといえ，プログラムに直接関係のないものや条件文の中身は指示した方がより注意深く見ることで発見できると考えられる．一方で，このパターンは画面表示の誤りや設計書と異なる初期値での初期化の検出に悪影響があるといえる．指示によって各ブロックの内容を完全に理解するまで細部を確認しなくなることで，細部の確認が必要なバグの発見が難しくなると考えられる．

図 4.10 のパターンは，Caller-Callee 関係にあるメソッド (func1,func3) のソースコードを繰り返し行き来した後に，周辺のメソッド (func2) へ上下に移動している視線である．このパターンは，B_1 と B_5 の検出率を増加させているのに対して，B_3 と B_4 の検出率を減少させているといえる．このことから，このパターンの実施は画面表示の誤りや演算子の誤りの検出に効果的であるといえ，画面の表示内容や条件文の中身は指示した方がより詳細な内容まで見るため，発見できると考えられる．一方で，このパターンは入力する変数の誤りの検出に悪影響があるといえる．変数自体にバグがなく，プログラムを実行するまでバグと判断できないため，メソッド間の関係性に着目する指示によって発見しづらくなると考えられる．

5. 結論

本研究はコードレビューにおける指示ありと指示なしグループの視線データから頻出パターンを抽出し，頻出パターンの該当者率やバグの検出率との関係を調べることバグ発見におけるレビュー効率の向上に効果的または悪影響であるパターンを分析した．

指示ありグループと指示なしグループの各被験者の視線データを視線の時系列情報記号に変換し，頻出系列パターンマイニングの 1 つである cSPADE を用いて被験者の半数以上が該当するパターンを頻出パターンとして抽出した．その抽出結果の内，"OT"を 1 つ以上含むものや他の頻出パターンに内包できるパターン長の短いパターンを対象から除外し，類似するパターンをグループ化して分析した．それぞれのグループで抽出された頻出パターンについて，もう一方にどのくらい含まれているかとパターンを実施している該当者率を算出した．バグ発見におけるレビュー効率の向上に効果的, 悪影響なパターンを明らかにするために，頻出パターンの実施の有無と個々のバグの検出の有無を集計し，パターンを実施した被験者のバグ検出率が +50% 以上，-50% 以下のパターンを分析した．

分析の結果，指示ありグループと指示なしグループに共通するパターンとして，Caller-Callee 関係にあるメソッドのブロック間を連続して移動している視線や Code-Design 関係にあるブロック間の移動をした後に，ソースコード内を上下に行き来する視線が見られた．また，指示無しグループのみに見られたパターンとして UP,DW といった上下に視線を移動する遷移が確認された．頻出パターンと該当者率の差において，指示ありグループでは，メソッドの設計から始まり，遷移関係のあるブロックを順に移動する視線が該当者率の差が正の値に大きいこと，指示なしグループでは，メソッドのコードを始点に上から順に視線を移動させ，Caller-Callee 関係にあるメソッドを再び読むために移動する視線が該当者率の差が負の値に大きいことが分かった．このことから，指示をすることで遷移関係のあるブロック間を優先的に移動する視線が多くなるのに対して，指示をしないことによって，被験者が遷移関係を考えずブロックを順に上下などに移動する視線が多くなったといえる．

頻出パターンとバグ検出率の関係において，Caller-Callee 関係にあるメソッドの設計と対応するコードを順に見て，そのメソッドの周辺を上下に移動する視線パターンは，指示した状況において，不要な変数の宣言や条件文の誤りの検出は効果的であり，画面表示の誤りや設計書と異なる初期値での初期化の検出に悪影響であることが分かった．Caller-Callee 関係にあるコードのメソッドを繰り返し行き来した後に，前後のメソッドブロックを上下に移動する視線パターンでは，指示した状況において，画面表示の誤りや演算子の誤りの検出に効果的であり，入力する変数の誤りの検出には悪影響であることが分かった．このことから，レビューの指示を受けた場合にこれらのパターンを実施することによって，画面に表示されるものや文字の間違いに対しての検出，指示を受けなかった場合にはプログラムに直接エラーとして現れないバグに対しての検出が効果的であるといえる．

今後の課題として，すべての頻出パターンとバグ検出率との関係を調べて，バグ検出率の向上するパターンまたは低下するパターンを分析することや他のプログラムのコードレビューの視線データからの頻出パターンについても分析を行い，今回の分析結果のパターンと組み合わせて，議論することが必要である．明らかになった頻出パターンを利用することで，コードレビュー時の被験者の視線からレビューについてのアドバイスを表示する教育用のソフトウェアの開発に応用し，バグ検出率を向上できると考えている．

謝辞

はじめに，指導教員として研究において多大なるご指導をいただいた上野秀剛准教授に厚く御礼申し上げます．特にパターンの分析手法のところで様々な意見をいただき，参考にさせていただきました. また，査読教員である山口賢一教授には査読において様々な貴重な意見をいただき，厚く御礼申し上げます.

参考文献

[1] Mohammed.J.Zaki., “Spade:anefficient algorithm for mining frequent sequences,” *Machine Learning*, no. 40, pp. 31–60.

[2] 應治 沙織，上野 秀剛，“レビュー開始時における対象物の比較指示によるバグ発見率の向上”，電子情報通信学会　教育工学研究会，vol. 2015-01-ET，pp. 1–6，Jan. 2015.

[3] K. Y. Y. H. K. K. Takeshi.D.Itoh, Takatomi.Kubo, “Towards generation of visual attention map for source code,” *Proceedings of APSIPA Annual Summit and Conference 2019*, pp. 951–954, Dec. 2019.

[4] メタバース相談室，“アイトラッキング技術とは？仕組みや活用方法を徹底解説！”，https://xrcloud.jp/blog/articles/business/2550/，Nov. 25. 2022 参照.

[5] 杉山麿人, “統計的優位性を担保するパターンマイニング技術,” **オペレーションズ・リサーチ誌** *4* **月号** *2017* **年**, vol. 62, no. 4, pp. 226–232, Apr. 2017.

[6] 松原 裕貴，宮崎 純，藤澤 誠，天野 敏之，加藤 博一，“cc-paid: cpu キャッシュを有効利用した並列時系列パターンマイニングアルゴリズム”，情報処理学会論文誌データベース (TOD)，vol. 4，no. 2，pp. 88–100，Jul. 2011.

[7] 應治沙織，上野秀剛，“ソフトウェアレビューにおける読み方の教示によるレビュー効率の変化”，第 19 回電子情報通信学会関西支部学生会，p. 84，Feb. 2014.

[8] 武藤祐子，岡野浩三，楠本真二，“呼び出し関係を用いた単体テストおよび静的検査の可視化手法の改善とその評価”，pp. 163–168，Dec. 2011.