卒　業　研　究　報　告　書

令和2年度

研究題目

コードレビューの視線データの分析による
レビューパターンの傾向の違い

指導教員　　上　野　秀　剛　　准教授

氏　　名　　二　宮　凌　真

令和3年1月26日　提出

奈 良 工 業 高 等 専 門 学 校　　情 報 工 学 科

コードレビューの視線データの分析による レビューパターンの傾向の違い

上野研究室　二宮凌真

本研究の目的はコードレビューにおいてレビューに関する指示を受けて，レビューを行う場合と被験者自身が考えるレビュー方法で行う場合において,それぞれのレビューパターンにどのような傾向があり,その中からバグ発見への影響が大きいとされるパターンはどのようなものであるかを明らかにすることである．本研究では，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を用いて，PrefixSpanというパターンマイニング手法でそれぞれのグループにおける頻出パターンを抽出する．また，それらの頻出パターンの該当者を明らかにして，指示ありグループと指示なしグループの実施率の差を求める．実験の結果，指示ありグループにおける該当者の割合が多くなったのは，設計書の説明と物件の検索メソッドの呼び出し部分を交互に見るパターンであり，指示なしグループにおける該当者の割合が多くなったのは，メインクラスを含むソースコードと設計書に書かれているプログラムの大まかな流れを確認した後に，物件の検索メソッドを見て，サブクラスメソッドの説明を確認するパターンであった．また，バグ発見率と頻出パターンの実施率の関係を調べた結果，バグを見つけやすいパターンには，必要でない変数が使われているバグにおいてプログラム内のある変数が使われている場所を探索するパターンがあり，バグが見つけにくいパターンには，変数の定義が設計書と異なるというバグおいてソースコードに定義されいる変数の値と設計書に書かれいている変数の値を比較するというパターンであることが分かった．

目 次

1 はじめに **1**

2 準備 **3**

2.1 視線計測 3

2.2 時系列パターンマイニング 3

2.3 レビューにおける指示の有無 4

3 実験 **6**

3.1 分析対象 6

3.1.1 視線データ 6

3.1.2 レビュー対象プログラム 7

3.2 分析手順 11

4 結果と考察 **13**

4.1 指示ありと指示なしの頻出パターンの比較 13

4.2 バグ発見率と頻出パターン 18

5 おわりに **22**

謝辞 **23**

参考文献 **24**

1 はじめに

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.

コードレビューに関する研究として, 應治は被験者を読み方を指示するグループと指示しないグループに分け,レビューを実施し,レビュー効率とバグの指摘精度を比較し, 指示するグループが指示した読み方を実施しているかどうかを確認するためにレビュー中の視線を計測した[1]. 指示するグループには被験者間の解釈に違いを起こさないように,レビューの具体的な手順を指示し, レビューを行った. その結果, 視線移動において, 指示ありグループはその後, 設計書とソースコードを交互に移動しながら読んでいること, 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把握せずにレビューを行っていたことが分かった. バグ発見率においては, 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. それぞれのグループにおける被験者のコードレビューの傾向を頻出パターン(複数の視線データが該当するパターン)として抽出した. 指示ありグループの頻出パターンでは, 読み方の指示を行ったことで変数やメソッドが設計書で指定通りに使用されていることの確認, 変数やメソッドが実際に使用されているソースコード上の場所の確認をプログラムの流れに沿って行わせることでプログラム理解を促進させるという影響を与えていることが分かった. しかし, 指示なしグループの頻出パターンがどのような影響を与えられているのか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なしグループの頻出パターンにもプログラムの理解に影響を与えるパターンが存在するのではないかと考え, 指示なしグループの頻出パターンを分析する. また, それらのパターンを実施した時の各バグの発見率の差を比較し, 分析する.

そこで，本研究では，指示有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注目して，それらの頻出パターンの中からバグ発見率に有効または有害なパターンを分析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．バグ発見に有効なパターンが定義できれば，レビュー効率やバグ発見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元データから該当者を出力する．その結果から，指示したことによって出現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，有害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発見率と頻出パターンの該当率との関係を調べ,バグ発見に有効または有害なパターンを分析する．

以下，2章では実験の準備について説明し，3章では実験で使用するデータや実験手順について説明し，4章に実験結果を述べ，考察を行い，5章ではまとめ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)[5]は人間の視線を計測して心理を読む技術のことである. この技術では,「いつ見ているか」,「どこを見ているのか」,「どのように見ているのか」という3つの要素をデータ化し, ユーザの視線を可視化する. 主要な計測方法[7]には, 専用のデバイスで眼球を撮影することによって眼球の動きを追い, 視線を推測するという方法がある. 計測して得られる視線データにはユーザが何を見ているのか, どのくらいの時間を見ているのか, どのように視線が動いているかという情報が含まれている. 一方で, ユーザがある時間においてその領域を見ている理由を知ることはできない.

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[6]. その技術の一つに, 時系列パターンマイニング[2]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PrefixSpanである. このアルゴリズムは, 時系列データベース上で長さkの時系列パターンが出現する位置より後ろの範囲を投影時系列データベースとし, その範囲内のアイテムの支持度(長さk+1の部分列αの出現回数/投影時系列データの個数n)を調べ, あらかじめ設定した最小支持度よりも大きいアイテムを長さkの時系列パターンにつなげることで, 長さk+1の時系列パターンを発見する. 例として, 時系列パターンがA→B(パターン長が2)の時の時系列データベースを図1に示す.

		時系列データ					
		1	2	3	4	5	6
SID	S_1	A	C	B	C	A	D
	S_2	A	D	B	A	C	
	S_3	B	A	A	D	C	A
	S_4	C	C	D	A	B	A
	S_5	C	A	C	A	B	A

3個目のパターン候補	支持度
A	4
C	2
D	1
A,C	1
A,D	1
C,D	1

図 1 時系列パターンA→Bの時のPrefixSpanによるパターン探索

この図において”↓”よりも後ろの範囲が投影時系列データとなり，この範囲なアイテムの指示度を調べ，最小支持度を4としたときに，その値よりも大きいアイテム(A)をつなげ，パターン長3の時系列パターンとして発見する．

この処理を再帰的に行うことで，時系列パターンの長さが伸び，処理の進行とともに投影時系列データベースが小さくなるため，効率的に支持度の値を調べられる．また，投影時系列データベースは時系列パターンの出現位置を記録することにより得られる．

2.3 レビューにおける指示の有無

應治の研究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[4]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，バグ発見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を確認する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 実験

3.1 分析対象

3.1.1 視線データ

本研究では,過去の研究[1]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番号を割り当てた視線データを使用する.

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3.1.1に示す3点であり,レビュアーごとに読み方が異ならないように計測を行ったものである.

(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2 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, 指示の実施で, 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1, 表2に示す.

表1 各ファイルの内容

ファイル番号	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 (MAX-HOUSE) 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,図4,図5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に示す.

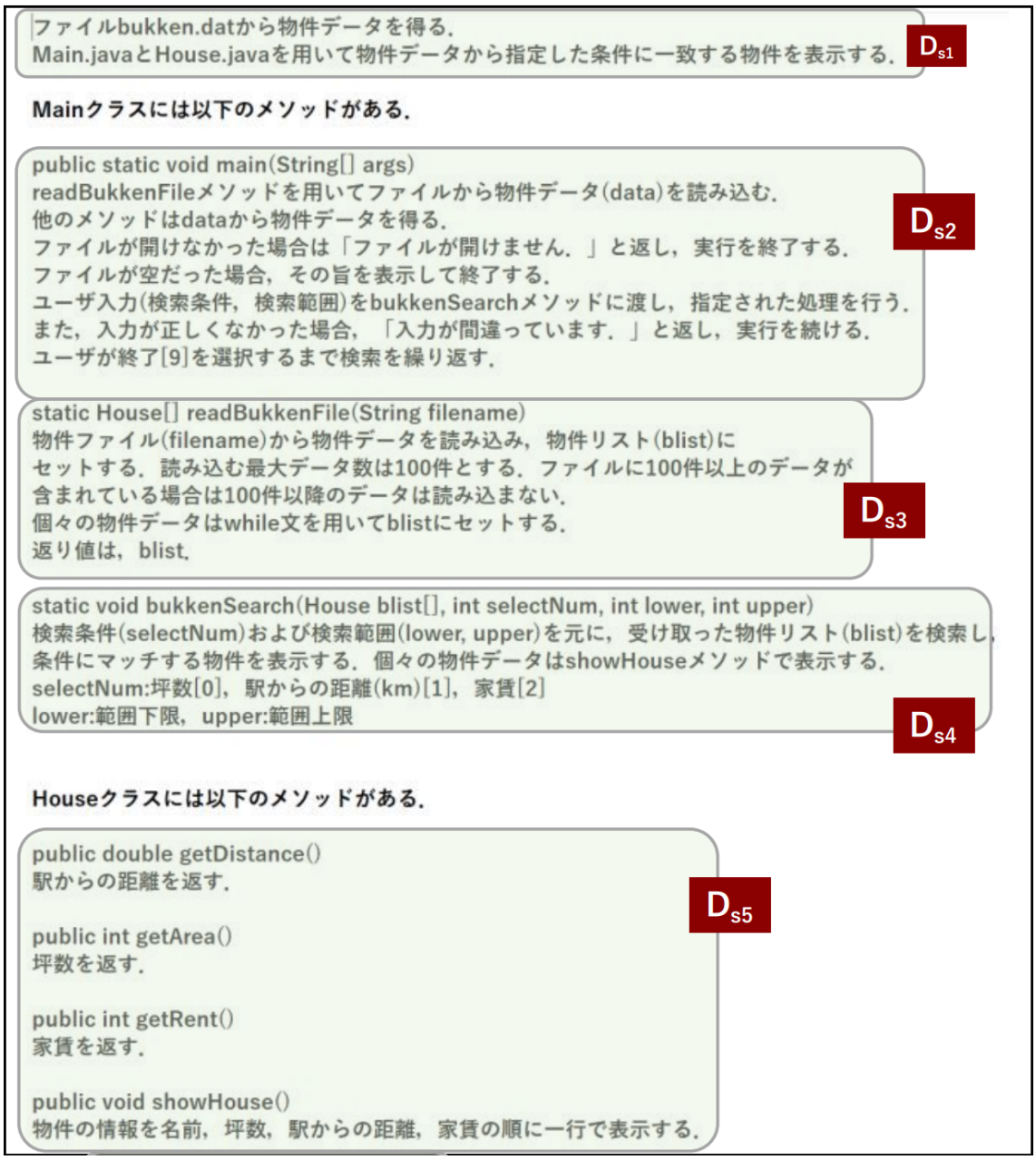

図3 設計書のブロック分割図

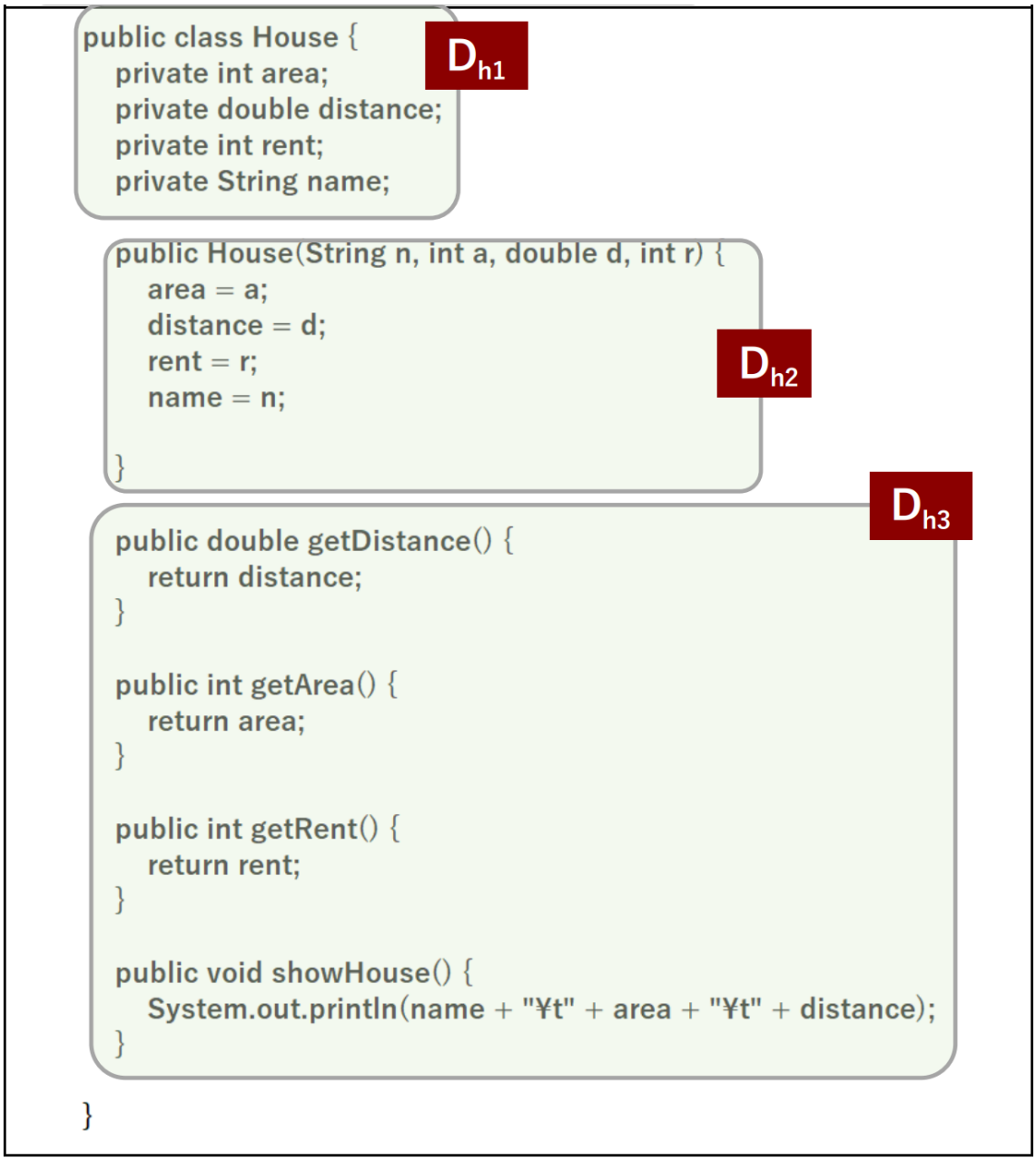

図 4 House.java のブロック分割図

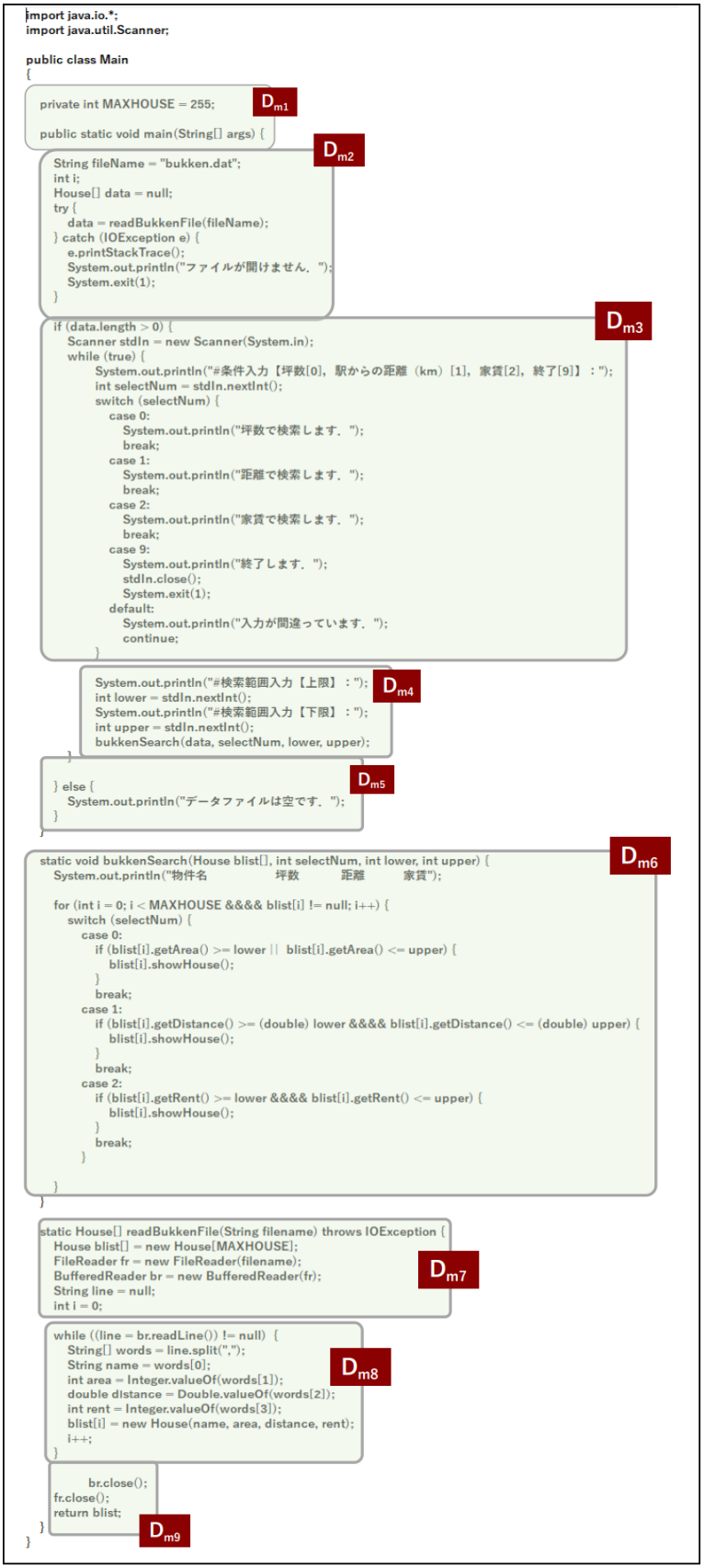

図 5 Main.java のブロック分割図

表 3 ファイルブロック分割の内容

ブロック番号	ファイル名	行番号	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

3.2 分析手順

分析の流れを表した図を図 6 に示す. まず, 指示ありの視線データと指示なしの視線データにおいてそれぞれのデータから頻出パターンを抽出するために, 2.2 節の PrefixSpan を用いて, 被験者の人数の内, 半数以上が該当するパターンを出力する. この時, 抽出するパターンの最長パターン長 max を 6, 最小パターン長 min

を1とした. 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，PrefixSpan を用いて，ある頻出パターンとある被験者の視線データで PrefixSpan を実行し，一致しているかどうかを出力する.

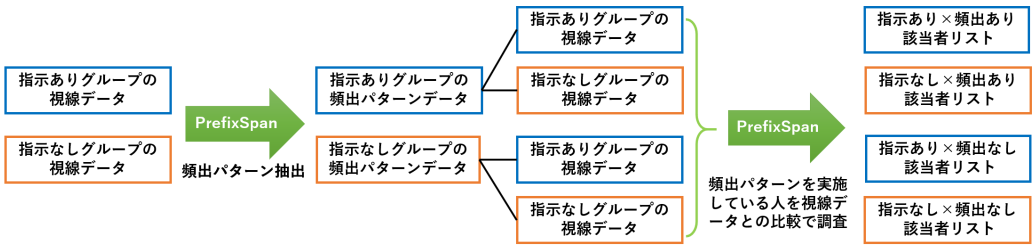

図 6 分析の流れ

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．指示あり各頻出パターンの内，指示ありグループと指示なしグループそれぞれの該当者率と指示なし各頻出パターンの内，指示ありグループと指示なしグループそれぞれの該当者率を調べるために，図7に示す4つの場合に基づいて分析する．

(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 7 パターン該当者を探索する場合分け

最後に，バグ発見率と各頻出パターンの指示ありグループと指示なしグループの該当者率の差を取ることで，バグ発見に繋がりやすい頻出パターン，バグを見つけにくいパターンを調べる．

4 結果と考察

4.1 指示ありと指示なし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PrefixSpanによる頻出パターンの抽出を行った結果，指示ありグループの頻出パターンの総数は398個，指示ありグループの頻出パターンの総数は2113個になった．

指示を行ったことで，増加したパターンと減少したパターンを調べるために，それらのパターンの中から，それぞれのグループにおける該当者の割合の差が30%以上であった頻出パターンを重点的に分析した．指示ありグループ，指示なしグループのそれぞれの被験者について指示ありにおける頻出パターンに該当者を調べ，指示ありグループでの該当者の割合と指示なしグループでの該当者の割合の差の絶対値が30%以上だったパターンを図8，図9，図10に示す．

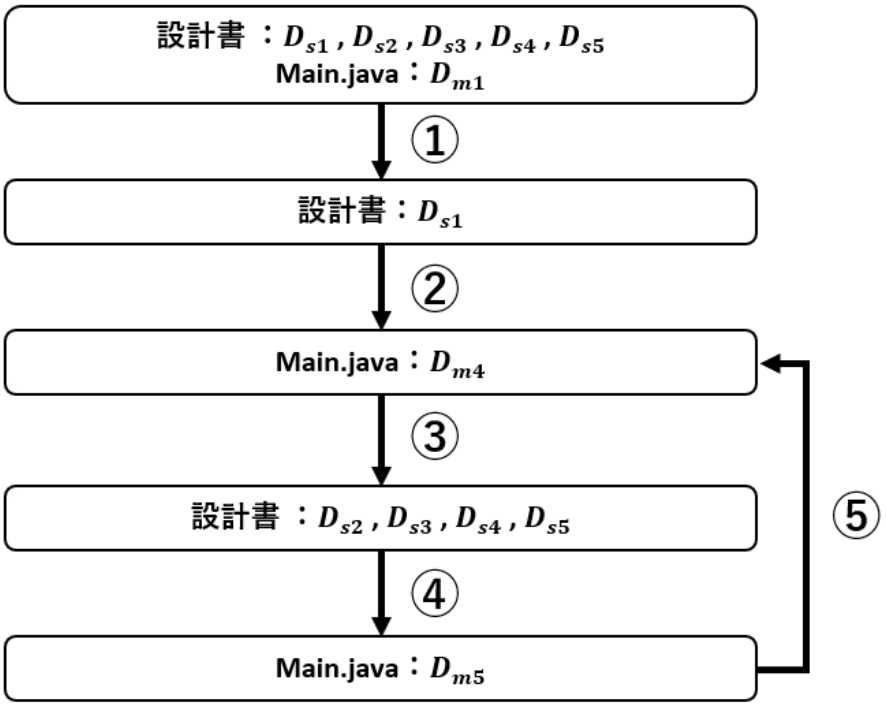

図8 変数の定義が異なるバグを探しているパターン

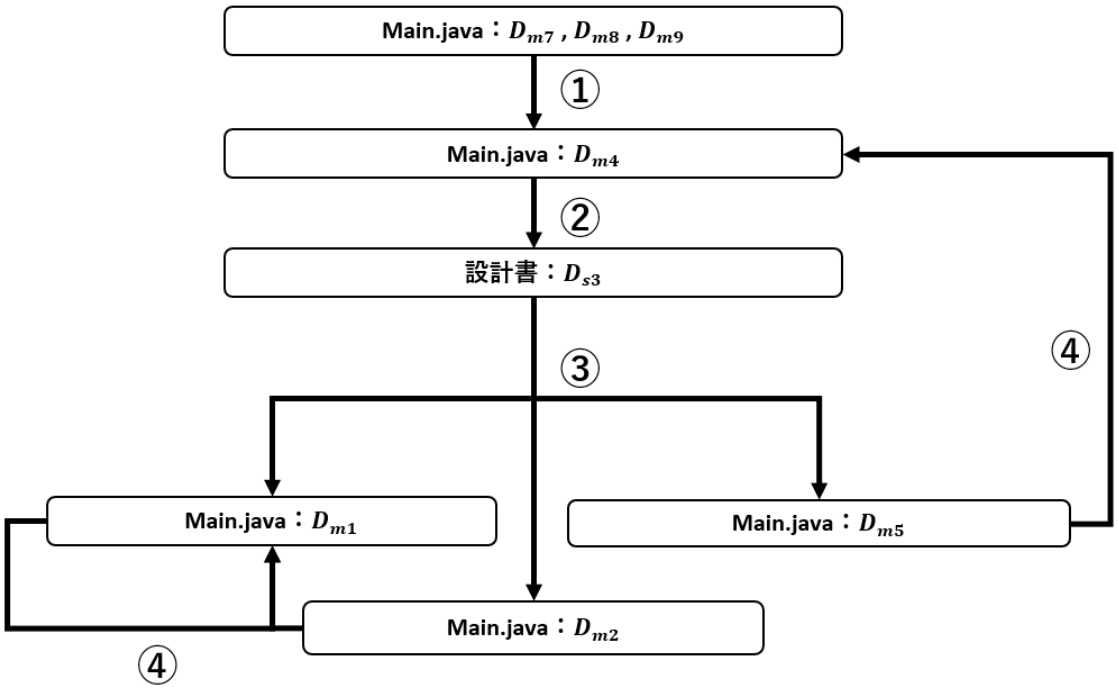

図 9 readBukkenFile についてとその関連情報を調べるパターン

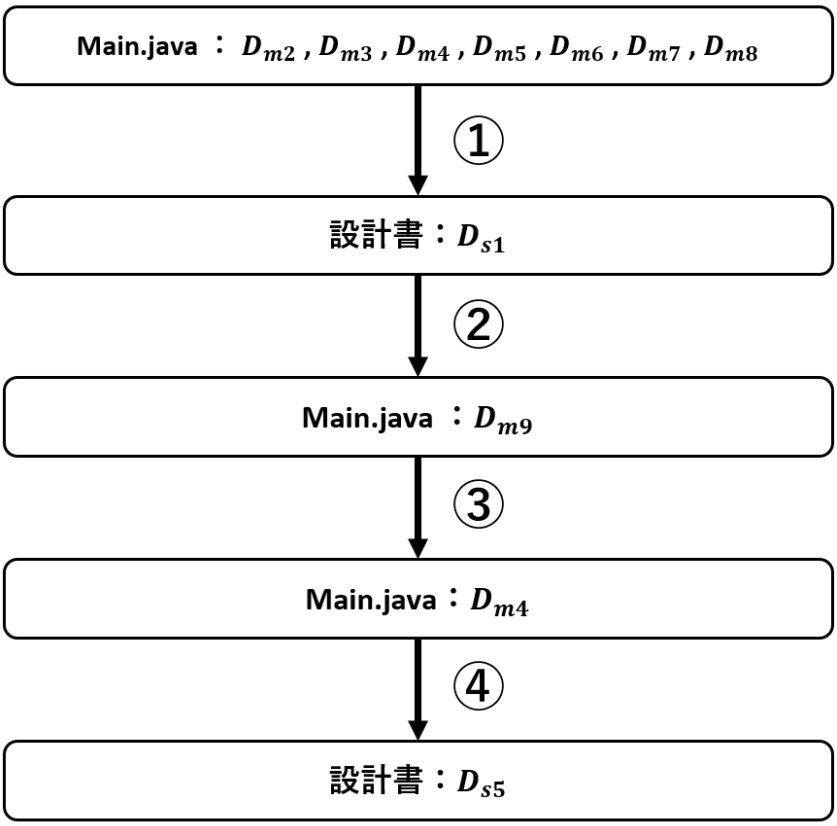

図 10 設計書の説明と物件検索メソッドの遷移を交互に見るパターン

各パターンを表す図内の番号は，視線の遷移を表しており，丸四角はそのときに注視していたブロックを表している．また，ブロックの内容は図3〜図5と表3を参照しながら説明する．

図8のパターンは，最初に設計書の各ブロックとMain.javaの最大物件数の定義の内の任意のブロックを見た(図3のD_{s1}〜D_{s5}か図5のD_{m1})後に，設計書の一番上にある大まかな流れ(D_{s1})を確認し，bukkenSearchのメソッドに使う検索数の条件(D_{m4})を見て，設計書のMain.javaの各メソッドの説明とHouse.javaのメソッド説明の内，任意のブロック(D_{s2}〜D_{s5})を確認し，物件ファイルが存在しないメッセージ(D_{m5})を確認した後，再び，bukkenSearchのメソッドに使う検索数の条件(D_{m5})という順に見ていることを表している．また，このパターンに該当する人数の割合の差が-38%で負の値で一番大きくなり，指示によって該当者が減少したパターンであることが分かった．このことから，物件配列の遷移先を理解して，関連の情報を調べながら，バグを探しているのではないかと考えられる．

図9のパターンは，Main.javaのreadBukkenFileの内容を見た(図5のD_{m7}〜D_{m9})後に，物件の検索数の設定及び検索処理メソッドの呼び出し部分(D_{m4})，設計書内のreadBukkenFileメソッドの説明(D_{s3})，最大物件数の定義やファイル存在確認の処理，ファイルが存在しないメッセージの表示の部分(D_{m1}，D_{m2}，D_{m5})を確認した後に，D_{m1}かD_{m5}という順に見ていることを表している．また，このパターンに該当する人数の割合の差が25%で正の値で一番大きくなり，指示によって該当者が増加したパターンであることが分かった．このことから，先にソースコードの中身を理解しようとするが，変数の意味やメソッドの内容を理解できず，最終的にメソッドの説明を見比べながらバグを探している移動であると考えられる．

図10のパターンは，最初にMain.javaの一番上もしくは一番下のブロック以外の内どれか1つを見た(図5のD_{m2}〜D_{m8})後に，設計書の一番上にあるプログラムの全体の流れ(D_{s1})を読み，Main.javaの一番下のブロック(D_{m9})を読み，そのブロックに含まれているbukkenlistが使われている関数の箇所(D_{m4})を確認して，設計書のHouse.javaの説明(D_{s5})という順に見ていることを表している．また，このパターンに該当する人数の割合の差が25%で正の値で一番大きくなり，指示によって該当者が増加したパターンであることが分かった．このことから，指示によって，設計書の内容を大まかに理解し，物件の検索処理に関わるメソッドを見たときに，変数の定義のバグに気づいたではないかと考えられる．

次に，指示なしグループの頻出パターンに対する指示ありグループでの該当者の割合と指示なしグループでの該当者の割合の差が大きかったパターンの一部をそれぞれ図11，図12，図13，図14に示す.

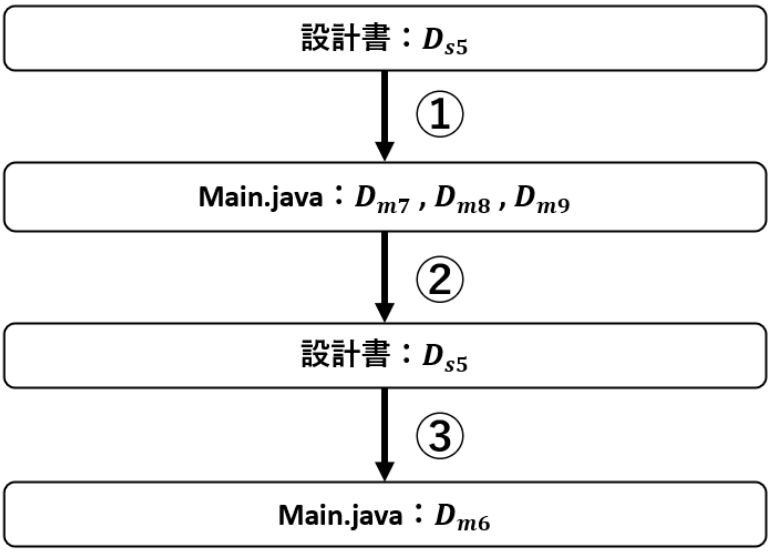

図 11 House クラスの変数と関連のあるメソッドを調べるパターン

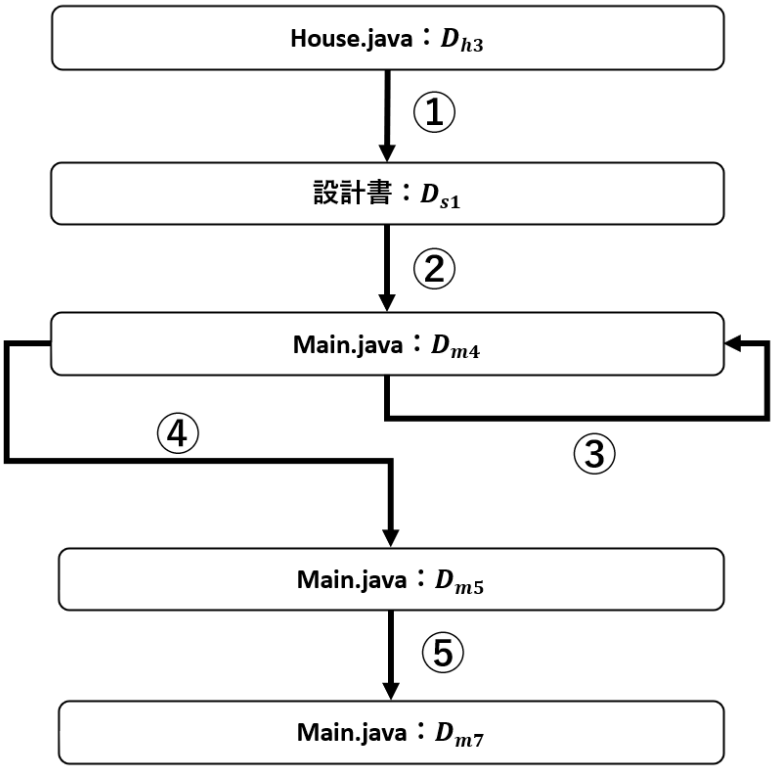

図 12 House クラスのメソッドから Main クラスのメソッドに確認するパターン

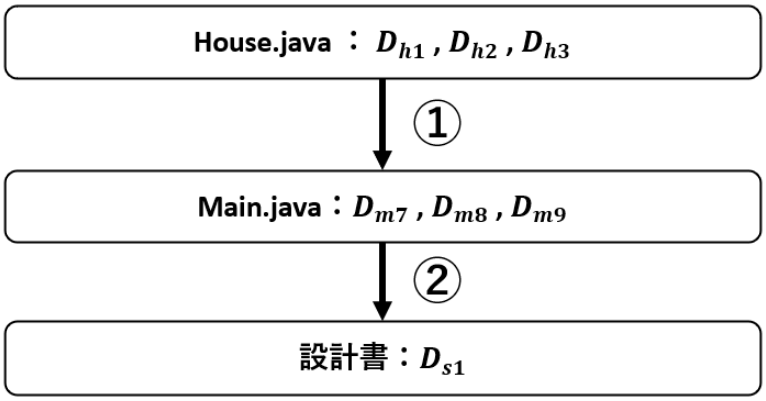

図 13 House クラスからプログラム全体の流れを読むパターン

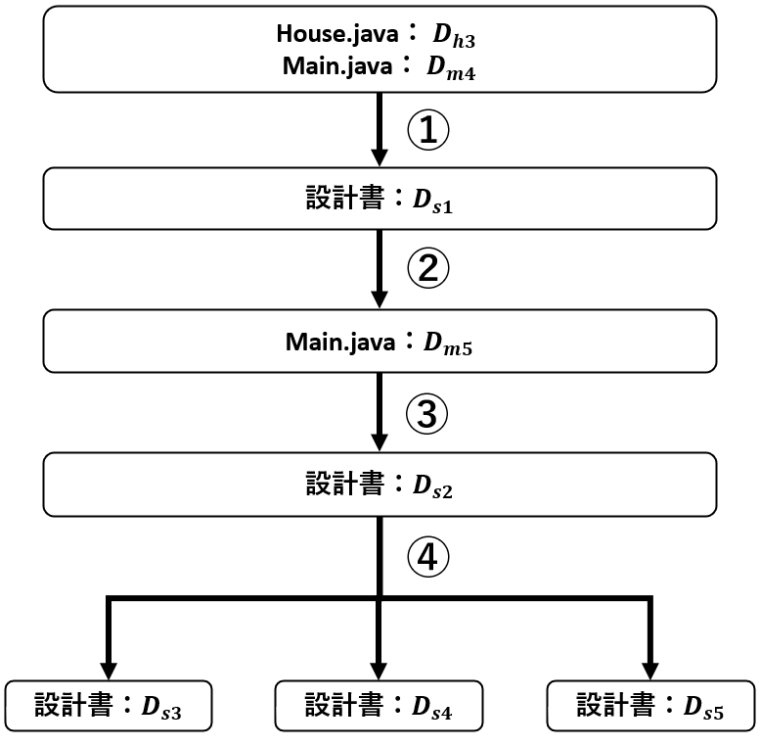

図 14 House クラス，Main クラスのメソッドから内容を理解するパターン

図 11 のパターンについて，House クラスの各メソッドの説明 (D_{s5}) を見た後に，bukkenSearch メソッドの説明の内の任意の 1 ブロック (D_{m7}〜D_{m9})，再び House クラスの各メソッドの説明 (D_{s5})，readBukkenFile メソッドの内容 (D_{m6}) という順に見ていることを表している．このことから，物件に関する情報を理解した後に，Main クラスのメソッドの中にバグがあるかどうかを探していたのではないかと考えられる．

図 12 のパターンについて，House.java の各メソッド (図 4 の D_{h3}) を見た後に，設計書の一番上にあるプログラムの処理の大まかな流れ (D_{s1})，bukkenSearch を呼び出す箇所 (D_{m4}) を見て，ファイルが存在しないメッセージを表示する部分 (D_{m5})，Main.java

のreadBukkenFileの変数定義の部分(D_{m7})という順に見ていることを表している．このことから，物件情報の取得方法を理解した後に，検索処理のメソッドと物件データの取得メソッドの中にバグを探していたのではないかと考えられる．

これらのパターンは指示をしないことによって，該当する人数の割合の差が20%であり，正の値に最も大きいものである．

図13のパターンについて，House.javaのクラス内変数の定義，コンストラクタ生成，各メソッドの内の任意の1ブロック(D_{h1}〜D_{h3})を見た後に，readBukkenFileの内容の内の任意の1ブロック(D_{m7}〜D_{m9})を確認し，設計書の一番上にあるプログラムの処理の大まかな流れ(D_{s1})という順に見ていることを表している．このことから，物件の情報を扱うクラスと物件データを格納するメソッドを確認してから，プログラムの全体の流れを理解して，バグを探そうとしているのではないかと考えられる．

図14のパターンについて，House.javaの各メソッドもしくはMain.javaのbukkenSearchの呼び出し箇所(D_{h3}，D_{m4})を見た後，設計書の一番上にあるプログラムの処理の大まかな流れ(D_{s1})，ファイルが存在しないメッセージの表示の部分(D_{m5})を確認した後に，Mainクラスのメインのメソッドの説明(D_{s2})を確認した.その後，設計書のMain.javaやHouse.javaのメソッドの説明(D_{s3}〜D_{s5})という順に見ていることを表している．このことから，物件情報に関するクラスとメインのクラスを理解した後に，物件検索処理と物件情報の格納メソッドにバグがないかどうかを探していたのではないかと考えられる．

これらのパターンは指示をしないことによって，該当する人数の割合の差が-13%であり，負の値に最も大きいものである．

全体として指示ありの頻出パターンと指示なしの頻出パターンの違いはあり，指示ありのパターンでは，設計書の内容を理解し，バグ発見のための情報を知った上で，バグの検出を行っているパターンが多かった．指示なしの頻出パターンには，ソースコードの内容を読みながら，バグの検出を行っているパターンが多かった．

4.2 バグ発見率と頻出パターン

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発見率を調べた結果，指示ありの頻出パターンではB_1とB_7の発見率が高く，B_2とB_5の発見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の発見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発見率との関係を調べ，その中でバグの発見率の差(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) の絶対値が50%以上となった一部のパターンをそれぞれ，図15，図16に示す．また，それぞれのパターンにおけるの各バグの発見率を表4に示す．

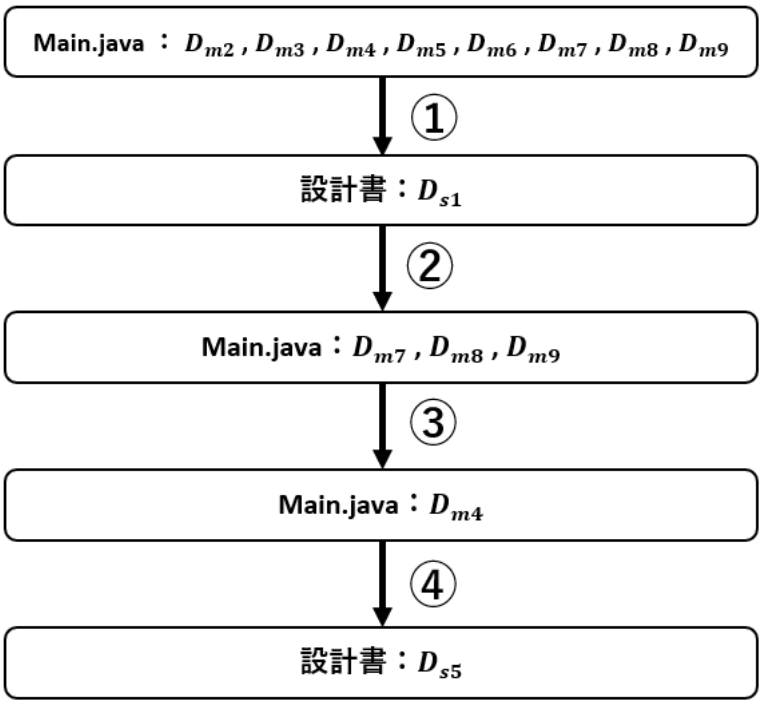

図 15 Main クラスのメソッドと House クラスのメソッドを理解するパターン

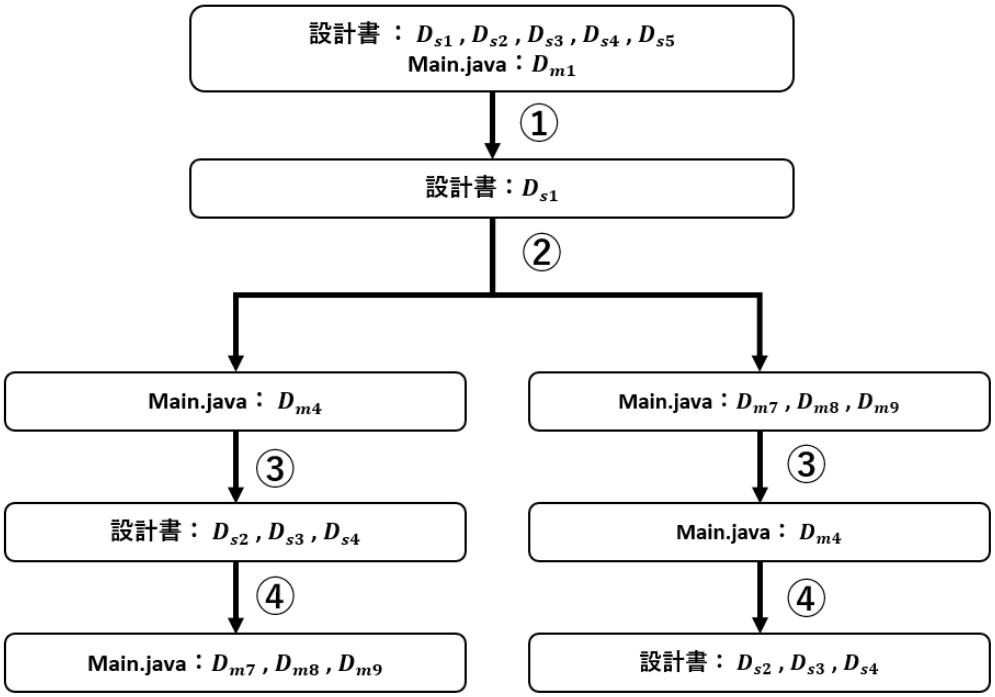

図 16 設計書から物件の検索処理に関するメソッドを理解するパターン

表4 図15と図16それぞれのパターンにおけるの各バグの発見率

バグ	パターン	
	図15	図16
B_1	38%	54%
B_2	-75%	-58%
B_3	42%	0%
B_4	42%	0%
B_5	-17%	-58%
B_6	75%	58%
B_7	-3%	-21%

図15のパターンは，Main.javaの一番上以外のブロックの内の任意の1ブロック(D_{m2}〜D_{m9})を見た後に，プログラムの全体の流れ(D_{s1})，返却される配列bukkenlistが含まれているreadBUkkenFileメソッドの内の任意の1ブロック(D_{m7}〜D_{m9})，検索数の入力やbukkensearchメソッドの呼び出し部分(D_{m4})を見た後に，Houseクラスの各メソッドの説明(D_{s5})という順に見ていることで，B_6の発見率を向上させているが，B_2の発見率を減少させていることがいえる．このことから，プログラムに直接関係のないバグの方が，指示しなくても見つけやすいが，数値の違いに関するバグは何度を確認しないと発見しにくいのではないかと考えられる．

図16のパターンは，設計書とMain.javaの最大物件数の定義のブロックの内，任意の1ブロック(D_{s1}〜D_{s5}，D_{m1})を見た後に，プログラムの全体の流れ(D_{s1})を理解し，返却される配列bukkenlistを見て，bukkenlistが使われているbukkenSearchメソッドの呼び出し部分(D_{m4})を見た後に，Mainクラスのメソッドの説明(D_{s2}〜D_{s4})を見た後に，readBukkenFileメソッド内の任意の1ブロック(D_{m7}〜D_{m9})という順に見ていることでB_1とB_6の発見率を向上させているが，B_2とB_5の発見率を減少させていることがいえる．また，設計書とMain.javaの最大物件数の定義のブロックの内，任意の1ブロック(D_{s1}〜D_{s5}，D_{m1})を見た後に，プログラムの全体の流(D_{s1})れを理解し，設計書におけるMainクラスのメソッドの説明の内の任意の1ブロック(D_{s2}〜D_{s4})を見た後に，bukkenlistが使われているbukkenSearchメソッドの呼び出し部分(D_{m4})を見て，返却される配列bukkenlistがあるreadBukkenFileメソッドの内の任意の1ブロック(D_{m7}〜D_{m9})という順に見ていることも同様のバグ発見率である．このことから，使われていない変数に関するバグはメソッドの内容を設計書の説明と交互に読み進めるため，見つけやすく，数値の違いに関するバグは最初に確認するだけであり，その後は間違っているかどうかを確認していないと考えられ，演算子の違いに関するバグは設計書の内容と比較するが，B_3とB_4を先に発見したこととメソッドの条件文についての確認するため，発見しにくいのではないかと考えられる．

それぞれの頻出パターン全体の結果は，バグの発見率と頻出パターンに関係はあり，指示ありの頻出パターンでは，Mainクラスと Houseクラスのソースコードを見た後にプログラムの流れを確認することで，変数の定義や不要な変数に関するバグが見つけやすくなっているのではないかと考えられる．指示なしの頻出パターンでは，物件情報が格納されている配列について検索処理メソッドや物件ファイルの取得処理のメソッドを見ることで，変数の抜けのバグを発見しやすくなったが，if文の条件の内容に関するバグを発見しにくくなっているのではないかと考えられる．

5 おわりに

本研究で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を調べることで指示によって増加したり，減少するパターンを分析した．また，それらの頻出パターンの該当者とバグ発見率との関係を分析した．先行研究では，指示有りグループの頻出パターンには，途中で変数の定義を確認するために設計書の確認をはさんで後にプログラムの流れに遡るパターンとプログラムの流れを確認するのではなく，変数が定義通りに初期化・使用されているかを確認するパターンがあり，それらがプログラムの理解を促進させていることが分かった。

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た結果，指示ありグループでは398個の頻出パターン，指示なしグループでは2113個の頻出パターンがあることが分かった.

今回の分析で分かったことは指示ありグループの頻出パターンには，設計書の説明を注視したのちに，対応するコードを見るというパターンの該当者数が減少しているのに対して，メインのソースコードから読み始め，検索処理に関連するメソッドを見ているパターンの該当者数が増加している。このことから，指示の内容に従った読み方をする方が減少していることがいえる。

頻出パターンとバグ発見率の関係はメインのソースコードから読み始め，プログラム全体の流れに沿ってバグを探すパターンは不要な変数が使われているバグを発見しやすいが，変数に設計書で示されているものと異なる値が格納されているバグは発見しにくいことが分かった。この結果から，プログラムの動作に影響しないバグの方が，指示しなくても発見しやすいが，変数の値が異なるバグは何度も確認しないと発見しにくいと考えられる．設計書から読み始め，物件の検索処理のブロックを見るパターンは変数が使われないバグと不要な変数を含むバグは見つけやすく，変数の値が設計書と異なるバグや条件文に演算子が異なるバグは発見しにくいことが分かった。この結果から，メソッドと設計書の説明を交互に読み進めるため，未使用の変数を見つけやすく，変数の数値の定義は最初に確認した後，異なるかどうかを確認していないと考えられ，演算子の違いは設計書の内容と比較するが，条件文を確認するため，発見しにくいと考えられる．

今後の課題として,すべての頻出パターンとバグ発見率との関係を調べて，バグ発見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，議論できるように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，バグ発見率を向上できると考えている．

謝辞

本研究を進めるに当たり, 上野秀剛准教授には指導教員として様々なご指導やご助言を頂き,深く感謝致します. また, 岩田大志准教授には査読教員として多数のご指摘を頂き, 心から感謝申し上げます.

参考文献

[1] 應治沙織，”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”，奈良工業高等専門学校専攻科電子情報工学専攻平成27年度専攻科特別研究論文，(2016).

[2] 松原裕貴，”時系列パターンマイニングアルゴリズムの高速化手法に関する研究”，奈良先端科学技術大学院大学，平成23年度奈良先端科学技術大学院大学情報科学研究科情報処理学専攻博士論文，(2012).

[3] Takeshi ITOH, Takatomi KUBO, Kiyoka IKEDA, Yuki MARUNO, Yoshiharu IKUTANI, Hideaki HATA, Kenichi MATSUMOTO and Kazushi IKEDA, ”Towards Generation of Visual Attention Map for Source Code”, Proceedings of APSIPA Annual Summit and Conference 2019, 2019.

[4] 應治沙織，”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”，奈良工業高等専門学校情報工学科平成25年度卒業研究論文，(2014).

[5] アイトラッキングとは|ユーザー視点で効果的なマーケティングを，KOTODORI, https://kotodori.jp/strategy/what-is-eye-tracking/, (参照：2020-11-25).

[6] 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4月号2017年，Vol.62, No.4, pp.226-232, (2017).

[7] 視線計測 eye tracking, UX TIMES, https://uxdaystokyo.com/articles/glossary/eye-tracking/ , (参照：2020-11-25).