卒　業　研　究　報　告　書

令和6年度

研究題目

プログラム理解時の構文要素に対する
注視特徴に基づいた効果的な読み

指導教員　　上　野　秀　剛　　准教授

氏　　名　　堀 川　　恭 吾

令和7年2月28日　提出

奈 良 工 業 高 等 専 門 学 校　　情 報 工 学 科

プログラム理解時の構文要素に対する注視特徴に基づいた効果的な読み

上野研究室　堀川恭吾

プログラム理解とは，開発者や研究者がプログラムの構造や動作，目的を効率的に把握するための方法を研究するソフトウェア工学の研究分野の1つである．プログラム理解における視線計測の研究はディスプレイ上の位置情報ベースで計測するものが多く，近年では構文ベースでの計測が登場し始めている．先行研究ではディスプレイ上の座標単位の視線移動を，構文木のノードに対する遷移に変換する手法が提案されたが，タスクの正誤と特定の構文要素との関係，および視線の遷移との関係は明らかになっていない．本研究ではディスプレイ上の座標単位で記録されたプログラム理解時の視線データを，先行研究の提案手法を用いて構文ノードに対する遷移に変換し，各構文要素に対する注視割合や視線の遷移と理解タスクの正否の因果関係を分析する．実験では各構文要素に対する注視量，制御フローに対する視線遷移パターン，複数メソッドの視線遷移パターン，タスク開始時，終了時の視線遷移パターンを正解／不正解ごとに比較し，多く見られた構文要素，視線遷移パターンを調べ，プログラム理解に効果的な読み方を分析した．視線の遷移の比較では，特定の構文要素に対する視線移動パターンから頻出パターンマイニングの1つであるcSPADEを用いて正解／不正解のグループにおける頻出パターンを抽出する．変数宣言部，出力部に対する注視量の分析の結果から，プログラム理解の成功者と失敗者で，低難易度タスクで変数宣言部に対する注視時間に差があることがわかった．一行に複数の変数宣言が含まれている場合に変数宣言部に着目しすぎない読み方をすることで，プログラム理解を効率的かつ効果的に進められる可能性がある．制御フローに対する視線遷移パターンの分析の結果から，プログラム理解の成功者と失敗者で，for文に対する視線の遷移パターンに差が見られた．for文に現れる初期化式，条件式，更新式の3要素すべてを見る読み方をすることで，プログラム理解を効率的かつ効果的に進められる可能性がある．複数メソッドの視線遷移パターンの分析の結果，プログラム理解の成功者と失敗者で，メソッド間に対する視線の遷移パターンに差が見られた．メソッドの呼び出しに沿った読み方，処理の流れを遡ってデータの流れを確認する読み方をすることでプログラム理解を効率的かつ効果的に進められる可能性がある．タスク開始時と終了時に見る視線遷移パターンの分析の結果，プログラム理解の成功者と失敗者で，タスク開始時と終了時の視線の遷移パターンに差があることがわかった．タスク開始時にアクセサメソッドを含む変数宣言に着目する読み方やタスク終了時に出力値に関する処理の流れを追う読み方をすることでプログラム理解を効率的かつ効果的に進められる可能性がある．

目 次

1 はじめに

ソフトウェア工学の研究分野の1つにプログラムを理解する過程を対象としたプログラム理解がある. プログラム理解とは, 開発者や研究者がソースコードを読み, プログラムの構造や動作, 目的を把握することである. プログラム理解の様子を分析することで開発者の思考プロセスやプログラムでの困難な箇所を特定し, コードの可読性向上や開発者の教育の改善が可能になる. 方法としてプログラム理解時の視線追跡や脳波などの生体情報の計測, 行動ログの記録などが挙げられる. プログラム理解の様子を分析することを目的に, 開発者がソースコードを読んでいる間の視線移動を計測し特徴を分析する研究が行われている [1][2][3].

プログラム理解における視線計測の研究はディスプレイ上の位置情報ベースで計測するものが多く, 近年では構文ベースでの計測が登場し始めている. 構文ベースでの計測をした先行研究として, Yoshiokaは視線計測装置が出力するディスプレイ上の座標単位の視線移動を, ソースコードから作成される構文木のノードに対する遷移に変換する手法を提案し, 構文要素の各種類に対する注視割合と理解タスクの正否との関係を分析した[4]. しかし, タスクの正誤と特定の構文要素との関係や視線の遷移との関係は明らかになっておらず, どの構文要素に着目して読むべきなのか, 次に着目すべき構文要素は何なのかなど, プログラム理解における効果的で効率的な読み方が不明瞭である.

本研究はプログラム理解の正誤とソースコードの構文要素に対する視線移動の因果関係を明らかにすることを目的とする. プログラム理解タスク中に計測した視線データを, 先行研究の提案手法を用いて構文ノードに対する遷移に変換し, 各構文要素に対する注視割合, 視線の遷移と理解タスクの正否の因果関係を分析する. 具体的には以下の4つの問い(Research Question)を立て, 分析を行う.

RQ1 プログラム理解の成功者と失敗者で各構文要素に対する注視時間に差があるのか?

RQ2 プログラム理解の成功者と失敗者で制御フローに対する視線の遷移パターンに差があるのか?

RQ3 プログラム理解の成功者と失敗者で各メソッドに対する視線の遷移パターンに差があるのか?

RQ4 プログラム理解の成功者と失敗者で理解開始直後と終了直前に見る視線の遷移パターンに差があるのか?

プログラム理解の正誤とソースコードの構文要素に対する視線移動の因果関係を明らかにすることで, プログラム理解のために着目すべき構文要素の種類や構文要素やメソッドの見るべき順番, プログラム理解開始と終了時の視線の移

動方法が明らかになり, 開発者に効率的な読み方を与え, 開発効率の向上や開発者の教育を促進する.

以下, 2章では関連研究について説明し, 3章で分析に関する準備について述べ, 4章で実験設定と手順を示す. 5章では実験の結果と考察を示し, 6章では本研究のまとめと今後の発展について説明する.

2 関連研究

YoshiokaとUwanoは視線計測装置が出力する座標単位の視線移動を，ソースコードから作成される構文木のノードに対する遷移に変換する手法を提案した[4]. 構文要素の各種類に対する注視割合と理解タスクの正否との関係を分析した結果，正解したタスクではifの条件式を有意に多く，メソッドの仮引数とprint文を有意に少なく見ていたことが確認された．また，再帰を含む複雑なタスクにおいてはif文の条件式よりもif文内の文をより多く見ていたことが分かった．本研究では，Yoshiokaらの研究で分析されていないタスクの正誤と特定の構文要素との関係や視線の遷移との関係を分析し，プログラム理解する上で効果的，効率的な読み方を明らかにする．

二宮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た[3]. 分析の結果，指示ありでは遷移関係のあるブロック間を移動する視線が多くなり，指示なしではブロックを順に上下などに移動する視線が多くなることが分かった．また，これらのパターンを実施することで，指示ありの場合には画面に表示されるものや文字の間違いに対しての検出，指示なしの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ことが明らかになった．この研究は，ブロック単位の頻出パターンに着目しており，本研究が行う構文ノードにおける細粒度での視線の頻出パターンは明らかになっていない．

大森らはより抽象的で理解が容易な変更履歴の生成を目指し，AST内部を含めた構文情報を保持可能な拡張操作履歴グラフFOHGを提案した[5]. 評価実験の結果，既存の手法よりも多様な自動リファクタリングを検出可能であること，構文要素に付加されるエッジを利用することで理解支援が可能であることを確認した．この研究はソースコードの編集時の操作履歴に着目しており，本研究が対象とする視線情報はより小さい粒度でプログラム理解の過程を見ることができる．

Rodegheroらはメソッド宣言，メソッドの呼び出し，制御フロー，その他の4種類の領域をソースコード上に定義し，プロのプログラマがソースコードを要約する視線を対象に，それぞれの領域に対する注視時間の長さを分析した[6]. 分析の結果，制御フローは，要約中に他のコード領域ほど頻繁には読まれておらず，メソッド宣言に重点を置いていることが明らかになった．この研究は，プロのプログラマの視線のみを対象としており，2群間の比較は行われていない．

3 準備

3.1 視線計測

視線計測(アイトラッキング)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[3]. 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，被験者の視線を装置のサンプリングレートごとに秒間数十から数千回のディスプレイ上の座標情報の配列として得ることができる. 装置から出力された座標情報を視覚化し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. 本研究では，被験者がプログラムタスクに取り組む際の視線をアイトラッカーを用いて計測し，プログラム理解時の視線の座標情報を得る.

3.2 Yoshiokaの提案手法[4]

Yoshiokaの提案手法では，視線計測装置が出力する座標単位の視線移動をソースコードから生成される構文木のノードに対応する遷移に変換する. 以下の図1にYoshiokaの提案手法の構成を示す. 四角はシステムを構成するモジュールを表し，細い矢印は情報の流れを表す. 視線計測装置は各時点における注視点のディスプレイ上の座標（例えば，X:121, Y:313）として時系列に出力する. 座標 - 行/列変換モジュールは視線計測装置から得られる座標単位の視線移動とソースコードを入力として受け取り，ソースコード名と行・列番号 (Main.java, 行:1, 列:13) として視線を出力する. このとき，行・列番号からソースコードの単語または文字を抽出し，構文解析で得られた構文木上のノードと対応をとる. また，同じ単語，文字に対する連続した注視は1つに統合する. 構文木/視線結合モジュールはパーサが生成した構文木と，行・列単位の視線移動を受け取り，構文ノード単位の視線移動(classDeclaration)を出力する. パーサが出力する構文解析の結果には，ソースコード中の各単語の位置を表す行・列番号と，文字数，およびその単語の構文種類が含まれる. 構文木/視線結合モジュールは行・列番号に変換された視線移動を構文解析結果の各ノードが持つ行・列番号と対応付けすることで，視線移動を構文木上のノード単位に変換する.

本研究では，被験者の視線を視線計測装置により計測し，得られた各時点における注視点のディスプレイ上の座標を提案手法を用いて変換し，各時点において被験者が注視した構文情報を分析する.

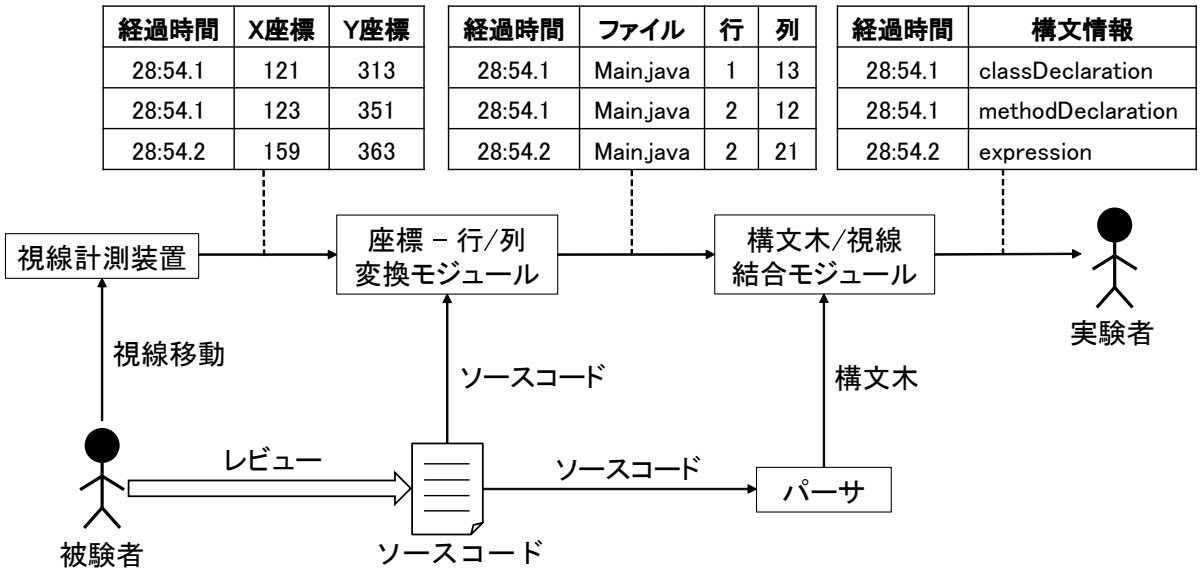

図 1: Yoshioka の提案手法の構成

3.3 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である [3]. その技術の1つに頻出パターンマイニングがあり, データ集合から出現頻度の高いパターンを頻出パターンとして発見する. 頻出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[7] である. このアルゴリズムは, 入力として表1のような系列データベースを得る. 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), 出現順序 (EID), アイテム (Item) 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3に示した. "A → B"というパターンは, "A" の後に"B" が出現するというパターンである. また, 支持度 (サポート) 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 1 の"A → B"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. 本研究では, 系列1つを1つの視線データとし, 被験者

表 1: 系列データベース

SID	EID(出現順序)	Items
1	1	A
	2	B
	3	C
	4	B
2	1	B
	2	A
3	1	B
4	1	C
	2	A
	3	B

表 2: 表 1 から作成される SID と EID をまとめたリスト

A		B		C	
SID	EID(出現順序)	SID	EID	SID	EID
1	1	1	2	1	3
2	2		4	4	1
4	2	2	1		
		3	1		
		4	3		

が見た構文要素を出現順序にならいアイテムとして頻出パターンを抽出する.

表 3: 表 2 系列パターン (A → B, C → B) に関するリストを結合してられるリスト

A→ B		C→ B		
SID	EID	SID	EID	
1	2	1	4	...
	4	4	3	
4	3			

4 実験

4.1 視線データ

石田の研究[2]で計測された視線データを使用する．この視線データは，日本語で記述されたプログラムの仕様とJavaで記述されたソースコードを被験者に提示し，処理内容を理解するタスクを与えた際の視線を計測したものである．被験者は奈良工業高等専門学校の学生14人で，年齢は19歳から21歳で，全員がJavaのプログラミング基本講義を受講済みである．実験は被験者1名と実験者2名のみがいる静かな部屋で実施する．実験では，視線計測装置，タスク提示用PC，記録用PCを使用し，視線計測にはTobii社製tobii Eye Tracker 4Cを用いる．この装置は，低コスト（200ドル未満）で非侵襲性があり，サンプリングレートが90Hzのスクリーンベースのアイトラッカーである．

被験者1人につき低難易度と高難易度それぞれ8タスクの計16タスクを与える．各タスクではプログラムの仕様とソースコードに加え，”6行目が2回目に実行された時のaの値を答えてください”のように，ソースコードの動作を理解できているか確認するための質問を提示し，回答内容が事前に用意した回答と一致した場合にソースコードの動作を理解したとみなす．回答が一致しない，あるいは制限時間を超過した場合，動作を理解していないとみなす．低難易度のタスクはmainメソッドのみからなり，1重の繰り返し文や条件分岐から構成される．高難易度のタスクは複数メソッドの使用や再帰構造を持ち，制限時間以内での理解が難しいと思われるソースコードを用いている．表4に実験で使用するタスクの仕様を示す．

タスクの制限時間は，低難易度タスクの理解には十分で，かつ，高難易度タスクの理解には不十分となる時間を予備実験により検証し，2分30秒と設定する．各タスクを提示する順番は，順序効果を考慮し，ラテン法各法によりカウンターバランスを行った．

4.2 分析

RQに回答するために表5に示すメトリクスについて分析する．

RQ1（プログラム理解の成功者と失敗者で各構文要素に対する注視時間に差があるのか?）に回答するために”重視する構文要素”を分析する．注視された構文要素のうち，変数宣言部と出力文に対する視線移動を対象に，注視時間（割合）を求める．注視割合は，特定の対象に向けられた注視時間の全体に対する割合であり，プログラムのどの部分にどれだけの認知資源を配分しているかを評価することができる．認知資源とは情報を処理したり，思考や判断を行ったりするために必要な脳の働きや精神的なエネルギーである．プログラム理解者の認知資源の配分が多い部分はプログラム理解を促進する部分と考えられ，プログラム理解

表 4: 実験で使用するタスク [2]

	タスク	仕様
1	Factorial	階乗の計算
2	SearchMax	最大値検索
3	PrimeNum	素数判定
4	SearchMedian	中央値検索
5	Power	累乗計算
6	Swap	2 つの数値の入れ替え
7	Substring	指定の文字列を含むか判定
8	ReverseString	文字列を反転させる
9	TowerOfHanoi	ハノイの塔
10	NumOfRoute	経路数を求める
11	Permutation	順列を全列挙する
12	Combination	組み合わせを漸化式から求める
13	PayMoney	支払う硬貨の組み合わせを求める
14	StrCombination	文字列の組み合わせを求める
15	CloudSim	雲の移動シミュレーション
16	lcm_gcd	最小公倍数と最大公約数を求める

に対して重要な箇所(構文要素)を知ることができる. そのため, 注視割合の計測は, プログラム理解における効率的かつ効果的読み方を明らかにすることができると考えた. 構文要素のうち, 変数宣言部と出力部を対象とする理由として, 変数宣言部はデータの初期設定となる部分, 出力部はプログラムの目的となる部分であり, プログラムの動作を理解する上で重要となる構文要素であり, 理解の有無で差が見られると考えたためである. 変数宣言部または出力文を表す構文要素への注視時間の合計時間の, タスク時間における割合を用いる.

RQ2 (プログラム理解の成功者と失敗者で制御フローに対する視線の遷移パターンに差があるのか?) に回答するために制御フローの一部である for 文を分析する. for 文に対する視線移動を対象に, for 文に出現する初期化式, 条件式, 更新

表 5: メトリクス

メトリクス	対象コード	調査対象
重視する構文要素	変数宣言部 出力部	注視時間 (割合)
制御フロー	for 文	見る順序
高い抽象度での要素	メソッド	見る順序
作業タイミングと構文要素	最初の 30 秒間 最後の 30 秒間	見る順序

式の3要素間における視線の遷移で見られた頻出パターンを抽出する．for文では，ループの開始条件を決める初期化式，ループの継続条件を決める条件式，各繰り返し後にループの状態を更新する更新式のループ処理における基本構造が記されている．そのため，制御フローの中でも，for文はループ処理の理解過程が視線に表れやすいと考え，初期化式，条件式，更新式を対象に視線の遷移パターンを分析し，成功者と不正解者間の差を調べる．頻出パターンを抽出するためにcSPADEを用い，各グループ（正解，不正解）に属する視線データのうち半数以上が該当する系列パターンとする．抽出するパターンの最長パターン長を10，最小パターン長を1とする．正解，不正解の被験者の間で差が見られたパターンとして，平均サポート値の差の絶対値が0.1以上であった視線移動パターンを用いる．また，正解，不正解の双方で平均サポート値が0.9以上のときはfor文を見るうえで当然のパターン，0.1以下のときはほとんど生じないパターンとして扱い，差が見られにくい視線移動パターンとして除外した．

RQ3（プログラム理解の成功者と失敗者で各メソッドに対する視線の遷移パターンに差があるのか?）に回答するために”高い抽象度での要素”を分析する．提案手法[4]ではより上位の構文要素で視線を要約することでブロックやメソッド，クラスなど高い抽象度の構文要素の遷移として見ることができる．高い抽象度の構文要素としてメソッドを対象に複数のメソッド間における視線の遷移で見られた頻出パターンを抽出する．メソッドは，プログラム内の特定の機能や処理をカプセル化した単位であり，メソッド間の視線移動を調べることは，データの流れの追跡やプログラムの全体構造の理解につながると考えられる．そのため，プログラムの理解の有無でメソッド間の視線の遷移が正解者と不正解者で異なると考えた．頻出パターンは，対象データを複数メソッド間の視線とし，RQ2と同様の方法で抽出する．

RQ4（プログラム理解の成功者と失敗者で理解開始直後と終了直前に見る視線の遷移パターンに差があるのか?）に回答するために”作業タイミングと構文要素”を分析する．最初の30秒間に対する分析では，タスク開始から30秒以内の視線で見られた遷移パターンで見られた頻出パターンを抽出する．最後の30秒間に対する分析では，タスク終了30秒前の視線で見られた遷移パターンで見られた頻出パターンを抽出する．全体を見てプログラムの目的や構造の把握をしてからソースコードを読み始めるのと，単純にソースコードを上から読み始めるのでは，プログラム理解の効率が違うと考え，タスク開始時の視線を分析する．また，ソースコードを読み終わる前に，全体の確認や重要部分の再確認などをすることでプログラム理解に影響があると考え，タスク終了時の視線を分析する．頻出パターンは，対象データをタスク開始から30秒以内の視線，タスク終了30秒前の視線とし，RQ2と同様の方法で抽出する．

表 6: 変数宣言部に対する注視量

task		宣言部（行数）	割合 (%) 正解	不正解	差
低難易度	1	1	8.5	16.7	-8.2
	2	2	19.0	17.7	1.3
	3	2	12.4	6.6	5.8
	4	1	13.7	0.0	13.7
	5	2	18.5	42.2	-23.7
	6	2	16.8	26.5	* -9.7
	7	5	33.7	0.0	33.7
	8	3	29.7	10.2	19.5
高難易度	9	0	-	-	-
	10	1	0.8	1.9	-1.1
	11	4	18.1	13.3	4.8
	12	2	7.6	8.7	-1.1
	13	3	9.0	11.0	-2.0
	14	2	10.0	6.9	3.1
	15	7	20.1	14.7	5.4
	16	0	-	-	-

* $p < 0.05$

5 結果と考察

256 件（16 人 × 16 タスク）のプログラム理解タスクのうち，計測誤差が発生したタスクを除外した，正解タスク 119 件，不正解タスク 81 件を分析する．

5.1 各構文要素に対する注視量

各構文要素に対する注視量の比較として変数宣言部と出力部に着目し，注視割合を求めた．表 6 に各タスクの変数宣言部の行数と，正解者と不正解者それぞれの変数宣言部に対する注視割合の平均を示す．また，task 9 と 16 で注視割合が”-”となっているのは宣言部がタスクのソースコード内に存在しないためである．変数宣言部は for 文中の初期化の式と，メソッド宣言部の仮引数の宣言部を除いた宣言部のみを対象とした．

低難易度のタスクでは task 3 と 8 で正解した被験者の注視割合が高く，task 1, 5, 6 では不正解の被験者の注視割合が高かった．このうち，task 6 のみで正解と不正解の平均値に有意差 ($p = 0.038$, t 検定) が見られた．task 8 では，String 型の変数から charAt メソッドを利用して部分文字列の抜き出しをしている．正解者はプログラム中の部分文字列の抜き出しをする処理を理解する際に，String 型変数の値に着目しており，変数宣言部への注視量の差に繋がったと考えられる．不正解の被

験者の注視割合が高かったtask 1, 5, 6はいずれも1行中に複数個の変数の宣言が行われており, 不正解者は, 複数の変数が1行中に宣言されているときは変数の型が行頭に1つしかないなどの理由から, 複数個の変数を同時に理解する際に, 正解者よりも多くの時間を要したと考えられる.

高難易度のタスクでは, task 11, 14, 15で正解した被験者の注視割合が高かった. task 14は低難易度のtask 8と同様にString型の変数からsubStringメソッドやcharAtメソッドを利用して部分文字列の抜き出しをしており, 正解者は変数宣言部に着目していると考えられる. task 11, 15は高難易度タスクのなかでも宣言部の行数が多いタスクであり, 宣言部の行数が多いときはプログラムの処理に直結する変数の初期値の要素を多く含んでいることから, 正解者は変数宣言部により着目してプログラム理解を進めていると考えられる.

表7に各タスクの出力部の行数と, 正解者と不正解者の出力部に対する注視割合の平均を示す. 低難易度のタスクではtask 1, 4, 5, 7で正解した被験者の注視割合が高かった. task 4, 7の出力部の行数はそれぞれ3, 4行と他の低難易度タスクと比べて多く, if文によって出力する内容が異なるプログラムである. if文により出力文が切り替わる場合, 実際に出力をする出力文を見る必要があり, 条件により出力文中の値が異なることから, 正解者は出力される文と値を判断する際に, 出力される値に着目してプログラム理解を進めていると考えられる. task 1, 5はそれぞれ階乗, 累乗を計算するプログラムであり, for文やwhile文の繰り返しの中で変数"a"の値を自己代入の累積により更新していく処理を含む. 累積を求めるプログラムでは, 自己代入している変数が累積をとることが多く, 出力となりやすいが, 正解者は出力文の出力される値に着目し, 何が出力されるのかを確認しており, 出力部への注視量の差に繋がったと考えられる. 高難易度のタスクでは, task 9 ,16で不正解した被験者の注視割合が高かった. task 9は2行にまたがる出力文が2つあり, 出力の内容は固定の文字列とメソッドの引数を結合した文字列となっている. 2つの出力文はそれぞれ1つ, 3つのメソッドの引数を含んでいることからtask 9の出力文は, 長く複雑な構造をしている. 高難易度のタスクにおいては, task 9の結果から, 出力文が複雑な構造をしており, 複数の出力パターンを持っているとき, 不正解者は出力の形式を理解する際に, 出力文の値に着目しており, 出力部への注視量の差に繋がったと考えられる. task 16は出力文の内容が引数を持つメソッドの呼び出し結果になっており, タスク不正解者はメソッド呼び出しにある引数の値とメソッドの戻り値との関係を理解する際に, 正解者よりも多くの時間を要したと考えられる.

各構文要素に対する注視量の分析から, RQ1（プログラム理解の成功者と失敗者で各構文要素に対する注視時間に差があるのか?）に回答する.

表 7: 出力部に対する注視量

task		出力部（行数）	割合(%) 正解	不正解	差
低難易度	1	1	4.4	0.2	4.2
	2	1	2.1	4.5	-2.4
	3	1	0.6	1.7	-1.2
	4	3	4.8	0.0	4.8
	5	1	3.6	0.6	3.0
	6	1	1.6	1.5	0.1
	7	4	14.1	0.0	14.1
	8	1	5.1	5.5	-0.4
高難易度	9	4	0.0	16.4	-16.4
	10	2	8.6	7.4	1.2
	11	2	1.8	1.8	0.0
	12	1	2.3	1.5	0.8
	13	1	1.9	2.0	0.0
	14	1	5.0	4.3	0.7
	15	2	2.0	2.3	-0.2
	16	2	11.0	14.5	-3.5

RQ1 への回答
低難易度タスクの変数宣言部に対しては差が見られたが，正解と不正解のどちらで注視量が多いかはタスクによってばらつきがあった．有意差の見られたタスクでは，不正解者の注視割合が高く，一行中の複数個の変数を同時に理解する際に，正解者よりも多くの時間を要したと考えられ，一行に複数の変数宣言が含まれている場合に変数宣言部に着目しすぎない読み方をすることでプログラム理解を効率的かつ効果的に進められる可能性がある．

5.2 制御フローに対する視線遷移パターン

制御フローに対する視線遷移パターンの比較としてfor文に出現する初期化式，条件式，更新式に着目して頻出パターンマイニングを行った．表8にすべてのタスクを通じて正解者，不正解者それぞれに見られた視線移動パターンとサポート値の平均を示す．

サポート値の差は正解のサポート値から不正解のサポート値を引いた値であり，差が正数のときに不正解よりも正解のサポート値が大きく，差が負数のときに不正解よりも正解のサポート値が小さいことを表している．

表8中の全ての視線移動パターンにおいて正解者と不正解者の間で有意差（$p<0.05$，カイ二乗検定）が認められた．表8の結果で3行目と4行目の視線移動パター

表 8: for 文中の制御フローに対する頻出パターン

Sequence	平均サポート値			カイ二乗検定	
	正解	不正解	差	$\chi^2(5)$	p 値
更新→ 条件	0.43	0.31	0.12	21.73	0.001
更新→ 条件→ 初期化	0.39	0.28	0.11	20.17	0.001
更新→ 条件→ 初期化→ 条件	0.38	0.26	0.12	18.52	0.002
初期化→ 更新	0.36	0.26	0.10	29.09	0.000
初期化→ 更新→ 条件→ 初期化→ 条件	0.22	0.11	0.11	14.53	0.013
更新→ 初期化→ 条件	0.30	0.19	0.11	40.47	0.000
条件→ 初期化→ 条件	0.79	0.94	-0.15	18.02	0.003
初期化→ 条件→ 初期化→ 条件	0.78	0.94	-0.17	17.58	0.004

ンは5行目の視線移動パターンに含まれており，6行目の視線移動パターンは7行目の視線移動パターンに含まれている．これらの視線をグループ化し，最も長い視線移動パターンをグループ内で見られた視線移動パターンの代表パターンとすると，正解者の視線で多く見られた視線移動パターンは以下に示す3つだった．

- 更新式→条件式→初期化式→条件式 （正解0.38, 不正解0.26）
- 初期化式→更新式→条件式→初期化式→条件式（正解0.22, 不正解0.11）
- 更新式→初期化式→条件式 （正解0.30, 不正解0.19）

一方で，不正解者の視線で多く見られた視線移動パターンは以下に示す2つだった．

- 条件式→初期化式→条件式（正解0.79, 不正解0.94）
- 初期化式→条件式→初期化式→条件式 （正解0.78, 不正解0.94）

正解者の視線でサポート値が0.1以上であったパターンはいずれも初期化式，条件式，更新式の3種類の要素すべてを含んでおり，不正解者の視線で多く見られたパターンは双方とも条件式と初期化式を交互に見るパターンだった．この結果から，正解者は，初期化式，条件式，更新式の3種類の要素に着目し，for文によるループ処理の流れを確認してプログラム理解を進めたと考えられる．

次に，初期化式，条件式，更新式の各要素単体に対するサポート値は，初期化式が正解0.98, 不正解1.00, 条件式が正解0.94, 不正解1.00, 更新式が正解0.59, 不正解0.54となっており，更新式に対する視線が少ない．初期化式を1つ以上含む視線移動パターン，条件式を1つ以上含む視線移動パターン，更新式を1つ以上含む視線移動パターンの平均サポート値を以下の表9に示す．表9より，いずれのパターンの平均サポート値に正解と不正解の被験者間の差はほとんど見られなかった．

表9: 初期化式，条件式，更新式の要素をそれぞれ1つ以上含む全視線移動パターンの平均サポート値

構文要素	平均サポート値		
	正解	不正解	差
初期化式	0.28	0.27	0.01
条件式	0.28	0.27	0.01
更新式	0.24	0.22	0.02

しかし，3つの構文要素を比較すると，更新式の平均サポート値が大きいことがわかる．

for文に対する視線遷移パターンの分析から，RQ2（プログラム理解の成功者と失敗者で制御フローに対する視線の遷移パターンに差があるのか?）に回答する．

RQ2への回答
プログラム理解の成功者と失敗者で，制御フロー対する視線の遷移パターンに差がある．for文に対して，初期化式，条件式，更新式の3要素をすべてを見る読み方をすることで，プログラム理解を効率的かつ効果的に進められる可能性がある．

5.3 複数メソッド間の視線遷移パターン

複数のメソッドを含むソースコードにおけるメソッド間の視線移動に着目し，頻出パターンマイニングを行った．メソッドが2つあるタスク（タスク12,13,14, mainとmethod1）と3つあるタスク（タスク9,10,11,16 mainとmethod1，method2）それぞれの視線移動の頻出パターンを求めた．

メソッドが2つあるタスク12,13,14のメソッドの呼び出し関係については，mainメソッドからmethod1が呼び出され，method1内で再帰的にmethod1が呼び出されている．頻出パターンマイニングの結果，最大パターン長が10までの全パターンにおいて，サポート値の差見られたパターンはmehod1とmainを交互に4〜5回見るパターンであり，その差は交互に4回見る場合，0.11（正解1.00, 不正解0.89），交互に5回見る場合は0.06（正解1.00, 不正解0.94）であった．その他のパターンでは差が見られなかった．正解者と不正解者の双方でサポート値が1に近い値をとっており，サポート値の差が小さいことから，いずれの視線もタスクの正誤に関わらず，ソースコードを見る上でよくみられる視線であると考えられるため，メソッドが2つあるソースコードにおいては，メソッド間の視線の移動はプログラム理解に与える影響がほとんどないといえる．

メソッドが3つあるタスク9, 10, 11, 16のメソッドの呼び出し関係を図2に示す．図中の”m1”がmethod1, ”m2”がmethod2を表している．図中の矢印が呼び出し関係を表

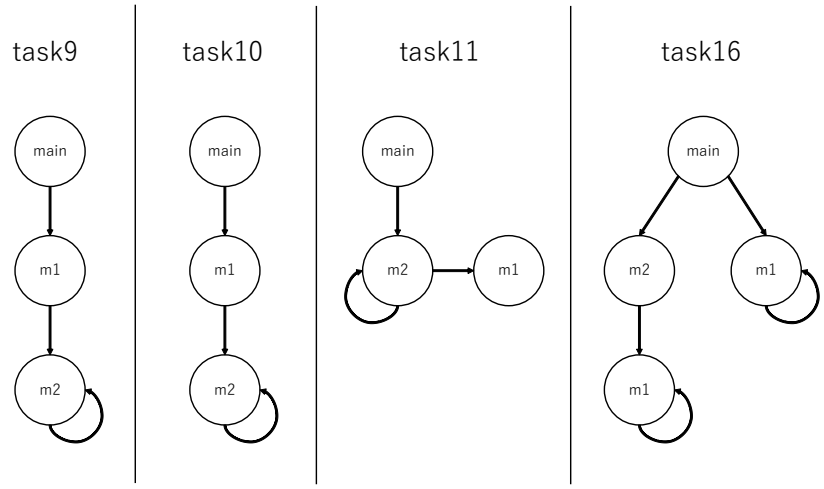

図 2: タスク 9, 10, 11, 16 のメソッドの呼び出し関係

しており, 矢印の方向にメソッドの呼び出しをしている. task9, 10 におけるメソッド呼び出しの関係については, main メソッドから method1 が呼び出され, method1 内で method2 が呼び出され, さらに method2 内で再帰的に method2 が呼び出されている. task11 におけるメソッド呼び出しの関係については, main メソッドから method2 が呼び出され, method2 内で再帰的に method2 が呼び出され, さらに条件分岐により method1 が呼び出されている. task16 におけるメソッド呼び出しの関係については, main メソッドから method1, method2 が呼び出され, method1 内では再帰的に method1 が呼び出されており, method2 内では method1 が呼び出されている.

表 10 に正解者, 不正解の被験者ごとのタスク中に見られたメソッドに対する視線移動パターンのうち, サポート値に 0.2 以上の差が見られたパターンとサポート値の平均を示す. ここでは, より特徴の見られたパターンとして, 平均サポート値の差がプラスの上位 5 件とマイナスの上位 5 件のパターンを示す. 5.2 節と同様に, 正解, 不正解の双方で平均サポート値が 0.9 以上, または, 0.1 以下のパターンを分析から除外した. また, 最も長い視線移動パターンをグループ内で見られた視線移動パターンの代表パターンとする.

表 10 の結果より, 正解の被験者の視線では, 表の 8, 10, 11, 12 行目のパターンでは main→ method2 のパターンを含んでおり, 不正解の被験者の視線移動ではみられていない. main→ method2 の視線移動パターンの平均サポート値は, 正解が 0.89 , 不正解が 0.67 であり, 差が 0.22 となった. カイ二乗検定により, main→ method2 の視線移動パターンは正解者と不正解者の間に 1% の水準で有意差が認められた ($\chi^2(3) = 11.84$,p <0.01). task 11, 16 では, main メソッドから metohd2 を呼び出しているため, main→ method2 の視線移動パターンは呼び出しに沿った視線移動と考えられる. メソッドの呼び出しに即した視線移動として, main→ method1 , method1→

表 10: 3メソッド間の頻出パターン

	Sequence	平均サポート値		
		正解	不正解	差
不正解者で多い	main → method1 → main → method1 → main	0.25	0.57	-0.32
	method2 → main → method1 → main → method1	0.05	0.36	-0.31
	main → method1 → method2 → main → method1	0.11	0.42	-0.31
	method1 → method2 → main → method1 → main	0.05	0.34	-0.29
	method2 → method1 → method2 → method1 → method2	0.48	0.70	-0.22
正解者で多い	method1 → main → method1 → main → method2	0.38	0.13	0.25
	method2 → method1 → method2 → method1 → main	0.84	0.54	0.31
	main → method2 → method1 → main → method1	0.48	0.15	0.32
	method1 → main → method2 → method1 → main	0.38	0.05	0.33
	main → method1 → main → method2 → method1	0.44	0.10	0.35

* $p < 0.05$

method2 , method2→ method1 の視線があるが, いずれの視線も隣接したメソッド間の視線であり, 多くの視線パターンに属するため, ソースコードを読む上で当然のパターンとなっていると考えられる.

表10の9, 10, 11行目のパターンではmethod2 → method1 → mainのパターンを含む視線が見られ, 不正解の被験者の視線移動ではみられていない. task9, 10の処理の流れでは, mainメソッドからmethod1を呼び出し, method1からmehtod2を呼び出す流れが含まれている. method2 → method1 → mainのパターンのパターンはtask9 , 10の処理の流れと反対向きに遷移する視線である. これより, 正解者は処理の流れを遡り, 引数などのデータの流れを把握してプログラム理解を進めたと考えらる.

また, 正解者と不正解者で差が見られたパターンでは, main → method1 → main など2つのメソッドを交互に見るパターンを含むパターンがあまり見られなかった. メソッドが2つあるタスクでの頻出パターン結果と表10の結果から, 2つのメソッドを交互に見る視線はプログラム理解に与える影響がほとんどないと考えられる.

制御フローに対する視線遷移パターンの分析から, RQ3 (プログラム理解の成功者と失敗者で各メソッドに対する視線の遷移パターンに差があるのか?) に回答する.

RQ3への回答
プログラム理解の成功者と失敗者で, メソッド間に対する視線の遷移パターンに差がある. メソッドの呼び出しに沿った読み方や処理の流れを遡り, データの流れを確認する読み方をすることで, プログラム理解を効率的かつ効果的に進められる可能性がある.

表 11: タスク開始後30秒の頻出パターン

	Sequence	平均サポート値		
		正解	不正解	差
不正解者で多い	プリミティブ型→ 変数宣言子ID	0.35	0.73	-0.38
	クラス宣言→ クラスまたはインターフェース修飾子	0.16	0.47	-0.31
	変数宣言子ID→ プリミティブ型	0.46	0.75	-0.29
	メソッド宣言→ 仮引数	0.10	0.32	-0.22
	整数リテラル→ 変数宣言子→ 整数リテラル	0.00	0.21	-0.21
正解者で多い	メソッド呼び出し→ 式→ メソッド呼び出し→ 式	0.16	0.00	0.16
	式→ 主式→ 式→ 主式	0.30	0.14	0.16
	ブロック文	0.55	0.38	0.17
	主式→ 式→ 主式→ 式	0.41	0.21	0.20
	文→ 主式→ 式	0.31	0.07	0.24

5.4 タスク開始・終了時の視線遷移パターン

タスクの開始直後と終了直前の時間帯における構文要素に対する視線遷移に着目し，頻出パターンマイニングを行った．表11にすべてのタスクを通じて正解者，不正解者それぞれのタスク開始から30秒後までに見られた視線移動パターンとサポート値の平均を示す．ここでは，より特徴の見られたパターンとして，平均サポート値の差がプラスの上位5件とマイナスの上位5件のパターンを示す．5.2節と同様に，正解，不正解の双方で平均サポート値が0.9以上，または，0.1以下のパターンを分析から除外した．また，最も長い視線移動パターンをグループ内で見られた視線移動パターンの代表パターンとする．

表11の結果より，表の3〜7行目より，不正解者はクラス，メソッド，変数の宣言を含む遷移パターンが正解者と比べて多い．本実験で用いたタスクのソースコードでは，ソースコード上部でクラス，mainメソッド，メンバ変数の宣言が行われているため，不正解者はタスク開始直後にソースコードを上から順に読み始めていると考えられる．一方で，正解者は”式”を含む遷移パターンが不正解者と比べて多い．8行目（メソッド呼び出し→ 式→ メソッド呼び出し→ 式）はメソッド呼び出しのメソッド名と式を交互に見ており．メソッド呼び出しを含む式に着目していると考えられる．12行目（文→ 主式→ 式）は演算やメソッド呼び出しを含む式に着目していると考えられる．8, 12行目の結果から，正解の被験者は，計算式とアクセサメソッドを含む変数宣言に着目しているといえる．また，9, 11行目の視線移動パターンは，”式”と”主式”の繰り返しであり，”主式”は式中の変数であることから，単に式に着目している視線移動パターンであるといえる．表11の10行目の視線移動パターンは”ブロック式”であり，これはブロックを表す．これらのことから，正解の被験者はタスク開始直後，アクセサメソッドを含む変数宣言に着目し，ブロックという高い抽象度で読み始めていると考えられる．

表12にすべてのタスクを通じて正解者，不正解者それぞれのタスク終了前30秒に見られた視線移動パターンとサポート値の平均を示す．表12の4〜7行目よ

表 12: タスク終了30秒前の頻出パターン

Sequence		平均サポート値		
		正解	不正解	差
不正解者で多い	変数宣言子ID → プリミティブ型	0.16	0.44	-0.28
	式 → 主式 → 式 → 文	0.00	0.24	-0.24
	主式 → 式 → 文	0.07	0.25	-0.19
	主式 → 式 → 主式 → 文 → 主式	0.00	0.18	-0.18
	主式 → 文 → 主式 → 文	0.00	0.18	-0.18
正解者で多い	型またはvoid	0.43	0.26	0.17
	メソッド呼び出し → 式 → 主式 → 式	0.24	0.07	0.17
	整数リテラル → 式	0.47	0.28	0.19
	主式 → 式 → 主式 → 式 → 主式 → 式	0.32	0.12	0.20
	式 → 整数リテラル	0.46	0.19	0.26

り, 不正解の被験者の視線では”式”, ”主式”, ”文”を含む視線移動が多く, 演算やメソッド呼び出しを含む式に着目していることがわかる. また, これらの視線移動パターンでの正解の被験者の平均サポート値はほとんど0であり, 正解の被験者の視線では見られなかった. これより, 表11の8,12行目の結果から, 終了時に演算やメソッド呼び出しを含む式に着目する読み方はプログラム理解において効率の悪い読み方であるといえる. 表12の8行目の視線は, ”型またはvoid”のみ含む視線移動パターンであり, 正解の被験者はメソッドの戻り値の型（void型）を確認する読み方をしていることがわかる. 表12の10, 12行目の視線移動パターンは, ”整数リテラル” と ”式” を含んでおり, 整数を含む式に着目している視線であると考えられる. 表12の9行目の視線は, ”主式”, ”式”, ”メソッド呼び出し” の3種類を含んでおり, これらはそれぞれ print 文の ”system”, ”out”, ”println” に当たる. そのため, 正解の被験者は出力文に着目する読み方をしたと考えられる. この視線移動パターンの不正解の被験者の平均サポート値はほとんど0であり, 不正解の被験者は出力文を見ていないことがわかる. これらの視線で着目された, 整数を含む式は変数の計算やメソッドの引数などで見られたため, 直接実行結果に関わりやすい要素であると考えられ, メソッドの戻り値の型（void型）は複数のメソッドを持つタスクにおいて, void型のメソッドは, 出力文を含んでいた. このことから, 正解の被験者はタスク終了30秒前では, 整数を含む式, 出力文に着目し, 出力値の処理の流れを追う視線移動をしていたと考えられる.

タスク開始・終了時の視線遷移パターンの分析から, RQ4（プログラム理解の成功者と失敗者で理解開始直後と終了直前にみる視線の遷移パターンに差があるのか?）に回答する.

RQ4 への回答
プログラム理解の成功者と失敗者で，タスク開始･終了時の視線の遷移パターンに差がある．理解開始直後にアクセサメソッドを含む変数宣言に着目し，ブロックという高い抽象度で全体構造を把握する読み方や理解終了直前に変数の計算やメソッドの引数に着目し，出力値の処理の流れを追う読み方をすることで，プログラム理解を効率的かつ効果的に進められる可能性がある．

6 おわりに

本研究はディスプレイ上の座標単位で記録されたプログラム理解時の視線データを，先行研究の提案手法を用いて構文ノードに対する遷移に変換し，各構文要素に対する注視割合や視線の遷移と理解タスクの正否の因果関係を分析した．構文要素に対する視線の遷移の比較では，頻出パターンマイニングの1つであるcSPADEを用いて，特定の構文要素に対する視線移動パターンを全パターン抽出し，分析をした．実験ではプログラムを理解する過程の視線データを対象に，各構文要素に対する注視量，制御フローに対する視線遷移パターン，複数メソッドの視線遷移パターン，タスク開始時，終了時の視線遷移パターンを正解者／不正解ごとに比較し，多く見られた構文要素，視線遷移パターンを調べた．

変数宣言部，出力部に対する注視量の分析の結果から，プログラム理解の成功者と失敗者で，低難易度タスクで変数宣言部に対する注視時間に差があることがわかり，一行に複数の変数宣言が含まれている場合に変数宣言部に着目しすぎない読み方などをすることで，プログラム理解を効率的かつ効果的に進められる可能性がある．制御フローに対する視線遷移パターンの分析の結果から，プログラム理解の成功者と失敗者で，for文に対する視線の遷移パターンに差が見られ，初期化式，条件式，更新式の3要素すべてを見る読み方をすることで，プログラム理解を効率的かつ効果的に進められる可能性がある．複数メソッドの視線遷移パターンの分析の結果，プログラム理解の成功者と失敗者で，メソッド間に対する視線の遷移パターンに差が見られ，データの流れを確認する読み方をすることでプログラム理解を効率的かつ効果的に進められる可能性がある．タスク開始時と終了時に見る視線遷移パターンの分析の結果，プログラム理解の成功者と失敗者で，タスク開始時と終了時の視線の遷移パターンに差があることがわかり，タスク開始時にアクセサメソッドを含む変数宣言に着目する読み方をすることでプログラム理解を効率的かつ効果的に進められる可能性がある．

今後の課題として，変数宣言部，出力部の注視について，変数名や型などのさらに小さな粒度での分析，あるいは注視回数と注視時間長の分析が挙げられる．変数宣言部，出力部の注視について，さらに小さな粒度で分析することで，宣言文や出力文の中でも着目するべき要素，着目しなくても影響の少ない要素がわかり，1文に対するプログラム理解における効率的かつ効果的な読み方が明らかになると考えられる．変数宣言部，出力部に対する注視回数と注視時間長の分析により，被験者が変数宣言部，出力部に対しどれだけ頻繁に注視しているかが明らかになると，変数宣言部，出力部のプログラム理解における重要度がわかる．

また，制御フローに対する視線遷移パターンについて，初期化式，条件式，更新式の3要素に，for文で繰り返し処理されるブロックを加えた4要素での視線遷移パターンでの計測・分析が挙げられる．for文の初期化式，条件式，更新式の3

要素での分析では，for文の動作に関する理解のみに焦点が当たっており，for文で繰り返し処理されるブロックを加えた4要素での分析することで，被験者が初期化式，条件式，更新式のどの要素が繰り返し処理されるブロックと関連付けて読んでいるのかかがわかり，ループ文全体での意味を理解する過程での効率的な読み方がわかると考えられる．

謝辞

本研究を進めるにあたり, 多大なるご指導を賜りました上野秀剛准教授に厚く御礼申し上げます. 特に, 分析方法にに関していただいた多くの貴重な意見が, 研究内容を深める助けとなりました. この場を借りて改めて御礼申し上げます. 査読教員の岩田大志准教授には, 本論文の作成にあたり, 貴重なご意見を賜りましたことに, 深く感謝申し上げます.

参考文献

[1] 曾我遼, 鹿糠秀行, 久保孝富, 石尾隆, ”視線と心拍を用いた理解状況の推定”, コンピュータソフトウェア, Vol.40, No.1, pp.24-44 (2023).

[2] 石田豊美, ”脳波と視線の同時計測によるプログラム理解状況の把握,” 奈良高専特別研究論文 (2020).

[3] 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,” 奈良高専特別研究論文 (2023).

[4] Y. Haruhiko and H. Uwano, ”An Analysis of Program Comprehension Process by Eye Movement Mapping to Syntax Trees,” Networking and Parallel/Distributed Computing Systems, Vol.18, pp.137-152, 2024.

[5] 大森隆行, 大西淳, ”拡張操作履歴グラフによるリファクタリングの理解支援”, コンピュータソフトウェア, Vol.40, No.1, pp.97-116 (2023).

[6] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D ’ Mello, “Improving automated source code summarization via an eye-tracking study of programmers,” in Proceedings of the 36th International Conference on Software Engineering (ICSE), pp.390-401 (2014).

[7] Mohammed.J.Zaki., “Spade:An Efficient Algorithm for mining frequent sequences,” Machine Learning, Vol.42, pp.31–60.