DEIM2022 F23-2

チーム開発のためのソースコード閲覧履歴可視化システム

石島菜々香† 高久 雅生††

† 筑波大学情報学群知識情報・図書館学類 〒 305–8550 茨城県つくば市春日 1-2
†† 筑波大学図書館情報メディア系 〒 305–8550 茨城県つくば市春日 1-2
E-mail: †s1811459@klis.tsukuba.ac.jp, ††masao@slis.tsukuba.ac.jp

あらまし 複数人の開発者によるチーム開発では，実装を行う前に他者の実装したプログラムを理解することが重要となる．しかし、ソースコードが膨大になると，プログラムの構造を理解し，実装予定の箇所に関連するコードを発見することは難しい．そこで，本研究では開発者間のソースコードに関する知見共有によってプログラム理解を支援するシステムを提案する．提案システムでは，ソースコードの閲覧履歴を記録し，ネットワークグラフによる可視化結果をユーザに提示することで関連コードの発見やコードリーディングに関する知見の共有を支援する．被験者実験の結果，本提案システムはクラスやメソッドの単純な存在の発見には一定の効果がみられたが，発見した関連性の意味を解釈するには，コードリーディング能力に類似した，閲覧履歴グラフを読解する能力が必要であることが示唆された．

キーワード プログラム理解支援，情報可視化，ネットワークグラフ，チーム開発

1 はじめに

1.1 研究背景

複数の開発者が関わるソフトウェア開発では，自分が実装を行う前に，他者が実装したプログラムを理解することが重要である．開発者がプログラムに変更を加えようとする場合，理解のプロセスでの変更予定のコードに関連する箇所の探索，他の開発者の実装意図の推測が必要となる [1]．そのため，開発者にとってプログラム理解は，コーディングに必要な能力の中で特に重要である．

一方，複数人での開発では，個人開発に比べてプログラムのソースコードが膨大になる．その中から適切な関連箇所を見つけ出すことは，対象プログラムに親しみのない開発者にとって容易ではない [2]．開発の生産性を上げるためには，開発者のプログラム理解の促進が重要である．

このような，ソフトウェア開発におけるプログラム理解支援の研究では 2 つのアプローチがとられている．

第一のアプローチは，プログラムの実行情報や静的解析結果を可視化することで，より高次の抽象的なプログラム理解を促すものである．プログラム理解支援の研究としては，このような可視化ツールの研究が特に盛んである [3] [4] [3]．一般に公開されている代表的な可視化ツールとしては，クラスやメソッドの依存関係グラフを提供する Sourcetrail [5]，関数の呼び出し関係や関連する変数の参照位置を一覧にする機能などを提供する Understand [6] が挙げられる．

第二のアプローチは，プログラムの構造に関するドキュメントの作成を支援することで，プログラム理解のハードルを下げるものである．このような研究としては Cheng ら [7] や Oezbek ら [8] の研究などに見られるドキュメント作成支援ツールがあげられる．

これら 2 つがプログラム理解支援の手法として主に研究されている．一方で，既存の研究には課題もある．可視化ツールの研究では，大抵がツールの対応言語が限定されている，または，ソースコードエディタとは別の独立したツールとして実装されている．Roehm らの研究 [9] によると，開発者はプログラム理解を目的とした専用ツールはあまり利用しないということが示唆されている．そのため，より開発者が利用しやすい形でのツールの提供が課題である．また，ドキュメント作成支援ツールの研究では，ドキュメントの作成には様々なツールが利用可能であるが，更新自体を完全に自動化することは出来ていない．Forward ら [10] や，Kotula [11] の研究によると，ソフトウェア開発のドキュメントは，作成後に時間が経ち，活用することができないほど古くなっている場合が多いことが明らかになっている．さらに，Singer らの研究 [1] によると，開発者はソースコードを読むのに比べてドキュメントはあまり参照したがらないということが明らかにされている．以上の事由から，プログラム理解支援の研究として様々な手法を用いたツールが開発されているが，実践的な場で継続的に活用されるものはごく少数に留まっている．

1.2 研究目的

本研究では，プログラム理解支援を目的としたシステムを提案する．その中で前述の 2 つのアプローチにおける課題を解決する．

まず，可視化ツールの課題について，システムをソースコードエディタの拡張機能として開発することで解決ができる．開発者にとって，エディタから離れることなくツールにアクセスできるかというのは，ツールの利用がコーディング作業を阻害しないためにとても重要である．

次に，ドキュメント作成支援ツールの課題について，他の開発者のソースコード閲覧履歴を記録・可視化することで解決ができる．前述の通り，既存のプログラムを読んで理解する行為は，各々の開発者によって実装前に常に行われる．この読解の

過程の記録を可視化し，他の開発者に提示する．それにより，ドキュメント作成という形での情報の再構成をする必要がなくなり，更新作業が不要となる．よって，本研究ではソースコード閲覧履歴を記録・可視化するシステムを開発する．

また，ソースコード閲覧履歴の可視化には先行研究の課題解決だけでなく，対象プログラムに関する知識に基づくソースコードの手がかりを先に開発に参加した開発者から得ることができるという利点もある．Roehm らの研究 [9] によると，開発者がコードを読み始める場所は，開発者自身の経験に基づき直観的に選択しているということが明らかになっている．そのため，先に開発に参加した開発者のソースコード閲覧履歴をたどることで，後から開発に参加した開発者は先人の経験からプログラム理解の足掛かりを得ることができると考える．

1.3 本研究のリサーチクエスチョン

本研究では，新しく開発に参加する開発者のプログラム理解の支援を目的として，先に開発に参加している開発者のコードリーディング履歴を可視化するプログラム理解支援システムを構築し，その有効性を検証する．これまでに述べた背景・目的に基づき、本研究のリサーチクエスチョンを次のように設定する。

RQ1: 提案システムによる閲覧履歴の記録・可視化手法が適切か.
RQ2: 提案システムによる閲覧履歴の可視化が関連コードの発見を助けるか.
RQ3: 提案システムによる閲覧履歴の可視化がコード全体の構造の理解を助けるか.

2 関 連 研 究

プログラム理解支援の領域では，前述の通り主に 2 つのアプローチがとられている．

2.1 プログラム構造可視化ツール

鈴木らの研究 [4] では，プログラムから静的に得られる情報だけでなく，実行時に得られる関数の呼び出し履歴や変数値の変化などを可視化する SEIV というツールを提案している．SEIV は C 言語で書かれたプログラムに対応している．

また，一般に利用されているソースコード可視化ツールとして SourceTrail [5] が存在する．SourceTrail は C/C ++，Java, Python で書かれたプログラムの静的解析をもとに生成した，関数の依存関係を示すグラフとコードビューを組み合わせ，ソースコードの概要についての情報を提供している．

丸山らによる研究 [3] では，プログラムの構造を 3 次元の樹木型のオブジェクトによって表現する CodeForest という可視化ツールを提案している．これらの可視化結果に表示されている樹木はクラスに対応しており，枝はメソッドに対応している．また，CodeForest では，可視化結果の樹木の色や葉の枚数をユーザが変更した履歴を記録する．これにより，ユーザが行ったプログラム理解に関する行動を残しておくことで，過去の行動を振り返り，プログラムを理解することに役立つとしている．

2.2 ドキュメント作成支援ツール

Oezbek ら [8] による研究では，ソースコード中に閲覧順序と処理の説明を付与したタグをつけることで，ソースコードの構造を理解するために順番にソースコードを見て回るツアー式のドキュメントを生成する，JTourBus というドキュメント生成支援ツールを提案している．

Cheng ら [7] による研究では，ソースコードとドキュメントをリンクさせ，ソースコードとともにバージョン管理することで，対象のプログラムについての説明がどの程度文書化されているのか，また，ドキュメントの更新がされているかを測定することができる GeekyNote というドキュメント管理ツールを提案している．

2.3 ソースコード閲覧履歴を利用した研究

Busjahn ら [12] による研究では，ソースコード閲覧時のアイトラッキングのデータから，プログラミングの初心者と上級者ではどのようにソースコードの読み方が異なるかを分析した．その結果，上級者は初心者よりもソースコードを直線的に読まないことが明らかになっている．

また，Talsma ら [13] による研究では，シンタックスハイライトによるプログラム理解支援がどの程度の効果を及ぼすのか分析するために，アイトラッキング，インタビューを用いてユーザのコードリーディング行動を調査している．

ソースコードの閲覧履歴としては，以上のようなアイトラッキングのデータを分析するような研究が主に多く存在する．

2.4 本研究の位置づけ

前述の先行研究では，それぞれのアプローチでそれぞれの課題を抱えている．プログラムの構造を可視化するアプローチでは，対応言語が限られてしまう，初回のセットアップが複雑であるなど，導入のしやすさの上で課題がある．ドキュメントの作成支援をするアプローチでは，ドキュメントの陳腐化を意識し，更新率の可視化や，過去の記録の振り返り支援によって，対策を講じているが，根本的な解決は出来ていない．

本研究では，開発者間の知見共有によってプロクラム理解を支援するという目的に加え，これら先行研究の課題を解決する手法を提案する．

3 提 案 手 法

本章では提案システムの概要について述べる．

3.1 閲覧履歴の記録

本提案システムにおける閲覧履歴は，閲覧していたソースコードが含まれるプログラミング構成要素の階層構造の記録と定義する．

閲覧履歴を構成する最小単位をレコードと定義し，閲覧履歴は複数のレコードによって構成される．レコードは，ソースコードの閲覧位置におけるプログラミング構成要素の階層構造,

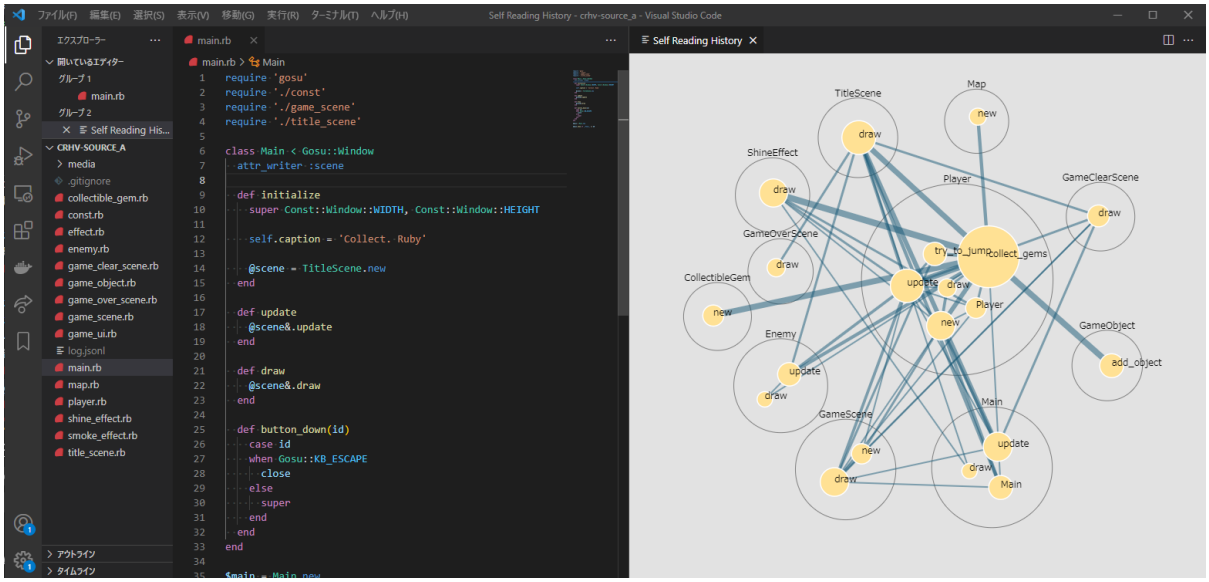


図 1: 提案システムのインターフェース

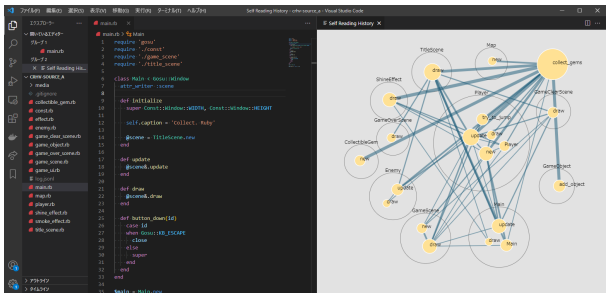


図 2: ドラッグ時のインタラクション

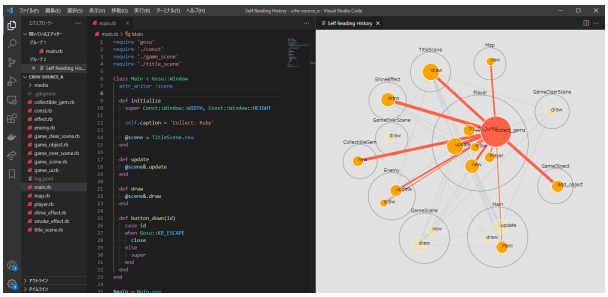


図 3: クリック時のインタラクション

閲覧したユーザーを示す UID，閲覧日時から構成される．本提案システムにおけるソースコードの閲覧位置 l_c は，ソースコードエディタ画面に表示されているソースコードの行番号の範囲 $[l_b, l_e]$ より，式 1 で表す．

$$l_c = l_b + \frac{l_e - l_b}{2} \tag{1}$$

また，レコードの生成は，閲覧位置のプログラミング構成要素の階層構造の変更を検知した時にされる．この変更の検知は一定間隔で行われ，エディタの画面がアクティブでない場合，レコードの取得休止を示すレコードを生成する．

3.2 閲覧履歴の可視化

本提案システムでは，Eduarda ら [14] のクラスタを持つノードの配置を最適化するアルゴリズムを適用した無向ネットワークグラフによって閲覧履歴の可視化を行う．この手法を使うことによって，ソースコードの階層構造とともにソースコードの閲覧箇所の関係性を可視化することが出来る．

グラフのノードはメソッドを示し，ノードを取り囲む黒い円はクラスを示している．青色のエッジはそのメソッドを示す閲覧履歴のレコードに前後関係がある (ただしレコードの取得休止を示すレコードは無視する) ことを示している．

ノード $n_i \in \mathcal{N}$ の半径 r_i は，そのメソッドの閲覧時間 t_i より，式 2 で表す．

$$r_i = \frac{\sqrt{t_i} - \sqrt{\min_{n_j \in \mathcal{N}} t_j}}{\sqrt{\max_{n_j \in \mathcal{N}} t_j} - \sqrt{\min_{n_j \in \mathcal{N}} t_j}} \times (30 - 10) + 10 \tag{2}$$

ここで，30，10 はそれぞれノードの最大半径，最小半径を示している．

エッジの幅は相互に参照した回数を示しており，ノード n_i，n_j 間のエッジ $e_{ij} \in \mathcal{E}$ の幅 w_{ij} はそれらメソッド間を遷移した回数 c_{ij} より，式 3 で表す．

$$w_{ij} = \frac{\sqrt{c_{ij}} - \sqrt{\min_{e_{kl} \in \mathcal{E}} c_{kl}}}{\sqrt{\max_{e_{kl} \in \mathcal{E}} c_{kl}} - \sqrt{\min_{e_{kl} \in \mathcal{E}} c_{kl}}} \times (20 - 2) + 2 \tag{3}$$

ここで，20，2 はそれぞれエッジの最大幅，最小幅を示している．

3.3 提案システムの利用方法

1 章で述べたように，本提案システムはソースコードエディタとの統合を目指す．その上で，本提案システムでは統合先のソースコードエディタとして，Visual Studio Code [15](以下 VSCode) を選択した．プログラミングに関する QA サイトである Stack Overflow が行った 2019 年の年次調査 [16] では，最も人気のあるエディタとして Microsoft 社が提供するコードエ

ディタである VSCode が挙げられている．よって，本研究では VSCode の拡張機能としてシステムを実装した．また，提案システムの閲覧履歴の記録，可視化結果の表示には VSCode Extension API [17] を，閲覧履歴の可視化には JavaScript 向けグラフ描画ライブラリである D3.js [18] を利用した．

VSCode 拡張機能は拡張機能のソースコードを vsce [19] を用いてパッケージ化したものである，VSIX ファイルを VSCode にインストールすることで利用可能になる．本提案システムも，提案システムの VSIX ファイルを VSCode にインストールすることで拡張機能として利用することができる．図 1 にシステムのインターフェースを示す．本提案システムでは，VSCode 上のコマンドパレットで選択可能な以下の 4 つのコマンドを備えている．

- 閲覧履歴の記録開始: crhv recording
- 閲覧履歴の記録停止: crhv stop recording
- 自身の閲覧履歴の可視化: crhv self visualization
- 他者の閲覧履歴の可視化: crhv others visualization

「crhv recording」コマンドを実行すると，記録開始を示すメッセージ表示とともに log.jsonl が生成され，閲覧履歴の記録が開始される．

「crhv stop recording」コマンドを実行すると，記録停止を示すメッセージ表示とともに閲覧履歴の記録が停止され，再び記録開始コマンドを実行するまで log.jsonl への書き込みは停止される．

「crhv self visualization」コマンドを実行すると，log.jsonl の内容をもとに生成したグラフを VSCode 上に表示することができる．

ユーザはグラフに対するインタラクションとしてドラッグとクリックの 2 つの操作が可能である．図 2，図 3 はそれぞれドラッグ，クリック時のグラフのインタラクションを示す．ユーザがグラフ上のノードをドラッグすることで，ノードの場所をずらすことができる．また，ユーザがノードをクリックすることでクリックしたノードと直接つながっているエッジとノードがハイライトされる．これらの機能は，グラフの文字や線が重なって見えずらい場合などに有効活用されることを想定している．

「crhv others visualization」コマンドを実行すると，log_others.jsonl の内容をもとに生成したグラフを VSCode 上に表示することができる．log_others.jsonl は事前に複数人の閲覧履歴を時系列順に結合して作成したファイルである．これは評価実験用に，他者の閲覧履歴をユーザ自身の閲覧履歴と混合させずに可視化する目的で作成した．本機能においても，ユーザ自身のグラフと同様のインタラクションを行うことが可能である．

4 評価実験

本章では提案システムの評価のために実施した被験者実験について述べる．

本実験では，1 章で提示した本研究のリサーチクエスチョンについて検証・評価を行うために被験者実験を行う．

4.1 実験方法

20 名の実験参加者を，実験参加者は実験日の日程調整の結果，前半 10 名をベースライングループ，後半 10 名を提案手法グループとして 2 つのグループに分けた．実験参加者には提案システムを利用しながら，課題プログラムに対して機能追加やバグ修正などの追加実装を行う課題に取り組んでもらう．また，課題前後にそれぞれ参加者のプログラミング経験，課題プログラムの自身の閲覧履歴から生成されたグラフについて，アンケートの回答を収集する．

課題前のアンケートでは，参加者の基本属性，プログラミング経験について調査した．

実験では，課題プログラムとして，デスクトップ上で動作するアクションゲームのソースコードを用意した [20]．このプログラムは Ruby とゲーム開発ライブラリの Gosu [21] を用いて実装している．出題した課題は 4 つで，前半 2 つが 10 分，後半 2 つが 20 分ずつの制限時間を設け，制限時間を超過した場合は，完成未完成に関わらず次の課題に取り組ませた．全ての課題は 1 行の変更または追加で完結するようにし，課題の回答を 1 行だけに制限した．ただし，デバッグのために途中で複数行の変更を行うことは許可している．また，課題への取り組みに際して，ベースライングループは提案システムによる閲覧履歴の記録機能だけを，提案手法グループは記録機能に加え，可視化機能を用いて提示された他者の閲覧履歴のグラフを利用する．他者の閲覧履歴とは，ベースライングループ 10 名分の閲覧履歴を統合して生成した閲覧履歴グラフを指す．図 4 に提案手法グループが利用する閲覧履歴グラフを示す．また，このグラフは可読性向上のため，グループ全体で 120 秒以上閲覧されているノードと 5 回以上遷移されているエッジのみという条件でフィルターをかけている．

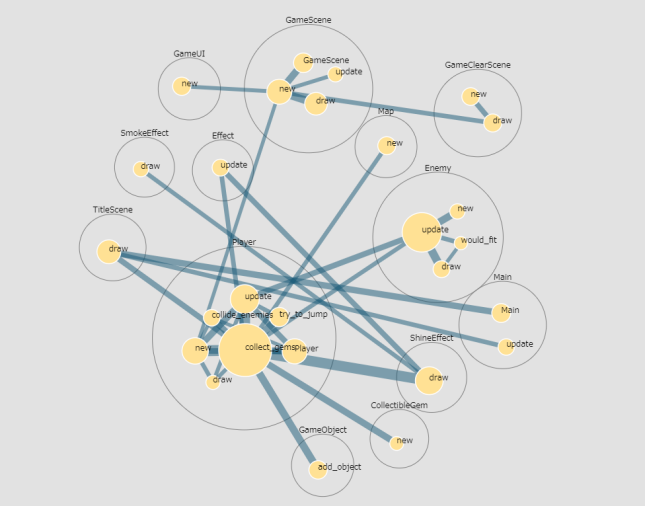


図 4: ベースライングループの閲覧履歴を統合して生成した閲覧履歴グラフ

課題後のアンケートでは，課題の理解度，記録した参加者自身の閲覧履歴グラフについての所感を調査した．また，提案手

法グループには他者の閲覧履歴グラフの利用についての所感も合わせて調査した．

5 結　　果

5.1 課題前のアンケートから得られた結果

5.1.1 プログラミング経験年数

参加者のプログラミング経験年数の手法別回答分布を図 5 に示す．ベースライングループは最大値 21，最小値 3，中央値 4.5，提案手法グループは最大値 12，最小値 2，中央値 3 という結果となった．また，各手法間でウィルコクソンの順位和検定を行ったところ，手法間で有意差が見られた ($p < 0.05$)．

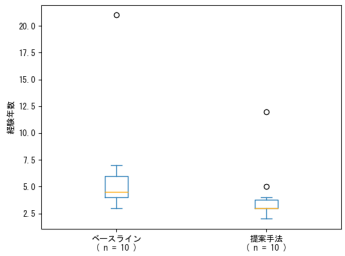


図 5: 参加者のプログラミング経験年数

5.1.2 プログラミング経験

参加者のプログラミング経験について当てはまるものを複数選択する設問における各参加者ごとの選択数を表 1 に示す．この設問では,「ライブラリの開発をしたことがある」,「オブジェクト指向を理解している」などプログラミングの経験，コードリーディングのテクニックについて 33 個の質問を列挙している．ベースライングループの回答数は提案手法グループと同数または上回る結果となった．

表 1: プログラミング経験についての質問の回答数

手法グループ	各参加者の回答数									
ベースライン	26	27	30	23	28	33	18	24	31	29
提案手法	12	20	28	16	9	13	18	31	26	24

5.2 課 題 結 果

5.2.1 課題正答率

実験課題ごとの正答率を図 6 に示す．全ての課題でベースライングループの正答率が提案手法グループを上回るという結果となった．

5.2.2 課題回答時間

課題 1 の回答時間の手法別分布を図 7a に示す．ベースライングループは最大値 240，最小値 60，中央値 107.5，提案手法グループは最大値 600，最小値 118，中央値 260 という結果となった．課題 2 の回答時間の手法別分布を図 7b に示す．ベースライングループは最大値 600，最小値 116，中央値 600，提案手法グループは最大値 600，最小値 175，中央値 600 という結果となった．課題 3 の回答時間の手法別分布を図 7c に示す．ベースライングループは最大値 1200，最小値 60，中央値 310.5，提案手法グループは最大値 1200，最小値 425，中央値 1185 という結果となった．課題 4 の回答時間の手法別分布を図 7d に示す．ベースライングループは最大値 1200，最小値 118，中央値 787.5，提案手法グループは最大値 1200，最小値 915，中央値 1200 という結果となった．また，課題 1〜4 における各手法間での回答時間に対してそれぞれウィルコクソンの順位和検定を行ったところ，課題 2 以外で有意差が見られた．

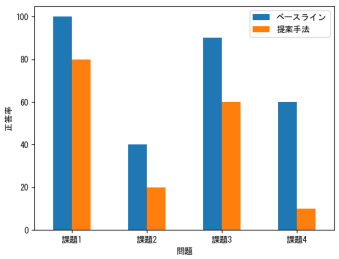
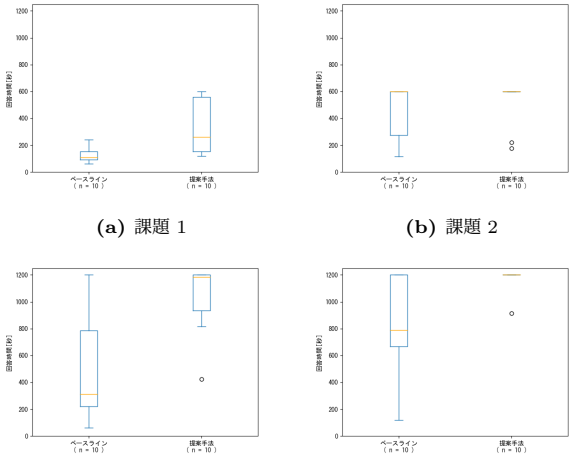


図 6: 参加者の課題ごとの正答率



(a) 課題 1　　**(b)** 課題 2

(c) 課題 3　　**(d)** 課題 4

図 7: 各課題の回答時間

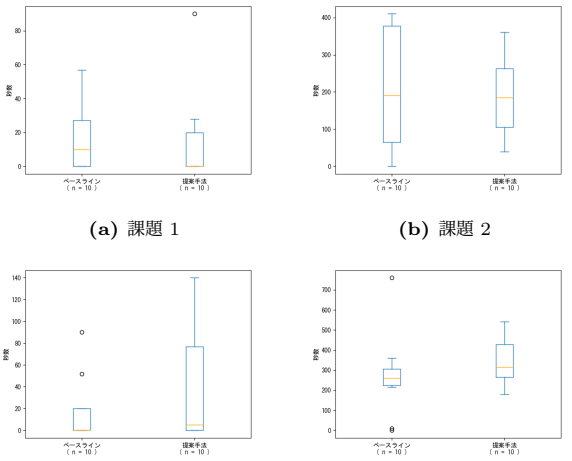
5.2.3 正解ノードの合計閲覧時間

各課題における変更を加えるべきメソッドを示すグラフ上のノードを正解ノードと定義する．制限時間内で正解ノードを閲覧していた時間の合計を各課題それぞれ述べる．

課題 1 における正解ノードの合計閲覧時間の手法別分布を図 8a に示す．ベースライングループでは最大値 57，最小値 0，中央値 10，提案手法グループでは最大値 40，最小値 0，中央値 0 となった．課題 2 における正解ノードの合計閲覧時間の手法別分布を図 8b に示す．ベースライングループでは最大値 57，最小値 0，中央値 10，提案手法グループでは最大値 360，最小値 40，中央値 185 となった．課題 3 における正解ノードの合計閲覧時間の手法別分布を図 8c に示す．ベースライングループでは最大値 90，最小値 0，中央値 0，提案手法グループでは最大値 140，最小値 0，中央値 5 となった．

課題 4 における正解ノードの合計閲覧時間の手法別分布を図

8d に示す．ベースライングループでは最大値 760，最小値 0，中央値 260，提案手法グループでは最大値 541，最小値 180，中央値 315.5 となった．また，課題 1〜4 いずれの場合でも，各手法間でのウィルコクソンの順位和検定の結果に有意差は見られなかった．



(a) 課題 1　(b) 課題 2

(c) 課題 3　(d) 課題 4

図 8: 各課題の正解ノード閲覧時間

5.3　課題後のアンケートから得られた結果

5.3.1　自身の閲覧履歴グラフ上のノードについて

課題後のアンケートで「自身の閲覧履歴グラフについて，閲覧した記憶はあるがグラフ上に表示されていないものがあるか」,「自身の閲覧履歴グラフについて，閲覧した記憶はないがグラフ上に表示されているものがあるか」について質問した結果を表 2，表 3 に示す．

表 2: 閲覧した記憶はあるがグラフ上にないもの，があるか

	ベースライン	提案手法
ない	6 人	7 人
分からない	1 人	2 人
ある	1 人	1 人

表 3: 閲覧した記憶はないがグラフ上にあるもの，があるか

	ベースライン	提案手法
ない	7 人	7 人
分からない	0 人	3 人
ある	3 人	0 人

記憶にあるがグラフ上に表示されていないもの，つまり，閲覧履歴からあぶれてしまっているものについて，そうしたものがないと答えた参加者の割合は 65 %であった．記憶にないがグラフ上に表示されているもの，つまり，閲覧履歴に不要なものがはいっているものについて，そうしたものがないと答えた参加者の割合は 70 %であった．

5.3.2　他者の閲覧履歴グラフが役に立ったか

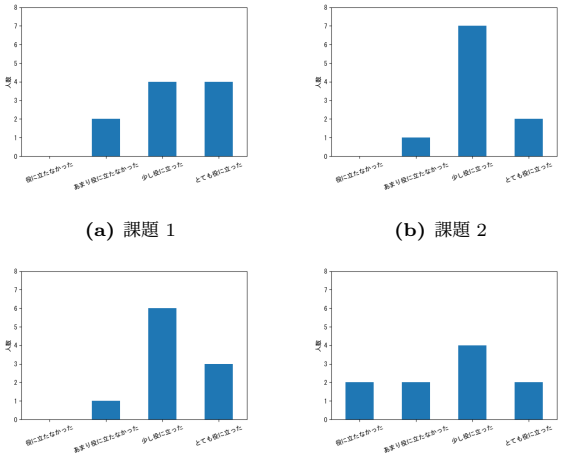
課題を解く際に他者の閲覧履歴グラフが役に立ったかどうかを提案手法グループのみに質問した．各課題ごとの結果を述べる．

課題 1 に対しての他者の閲覧履歴グラフの有用性についての回答総数を図 9a に示す．とても役に立った，少し役に立ったがどちらも 4 人，あまり役に立たなかったが 2 人，役に立たなかったが 0 人という結果になった．

次に，課題 2 に対しての他者の閲覧履歴グラフの有用性についての回答総数を図 9b に示す．とても役に立ったが 2 人，少し役に立ったが 7 人，あまり役に立たなかったが 1 人，役に立たなかったが 0 人という結果になった．

次に，課題 3 に対しての他者の閲覧履歴グラフの有用性についての回答総数を図 9c に示す．とても役に立ったが 3 人，少し役に立ったが 6 人，あまり役に立たなかったが 1 人，役に立たなかったが 0 人という結果になった．

次に，課題 4 に対しての他者の閲覧履歴グラフの有用性についての回答総数を図 9d に示す．とても役に立ったが 2 人，少し役に立ったが 4 人，あまり役に立たなかったが 2 人，役に立たなかったが 2 人という結果になった．



(a) 課題 1　(b) 課題 2

(c) 課題 3　(d) 課題 4

図 9: 他者の閲覧履歴グラフの有用性

5.3.3　グラフ表示の正確性

自身の閲覧履歴グラフについて，ノード，エッジ，グラフ全体に関する可視化結果が適切であるかについて質問した．各要素ごとの結果を述べる．

閲覧時間を反映したノードの大小が閲覧の記憶と一致していると思うか，という質問についての回答総数を図 10 に示す．ベースライングループで 1 名のみ一致していると思わないと回答した以外は全員が一致していると思うと回答した．

次に，メソッド間の遷移回数を反映したエッジの幅が閲覧の記憶と一致していると思うか，という質問についての回答総数を図 11 に示す．提案手法グループで 1 名のみ一致していると思わないと回答した以外は全員が一致していると思うと回答した．

次に，グラフ全体が閲覧の記憶と一致していると思うか，と

いう質問についての回答総数を図 12 に示す．両グループともにほとんど一致していたという回答が最も多く，また，全然一致していなかった，あまり一致していなかったとの回答はなかった．

これらの結果から，ノード半径とエッジ幅の大小の比率が正しいと感じた参加者の割合はそれぞれで 95 %であった．また，グラフ全体の表示に関しても，ほとんど一致していた，または，一致していたと感じた参加者の割合は 95 %であった．

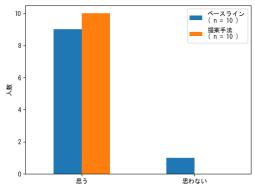


図 10: ノードの大小が記憶と一致すると思うか

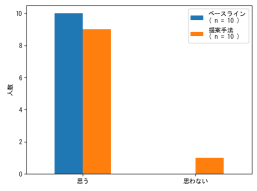


図 11: エッジの幅が記憶と一致すると思うか

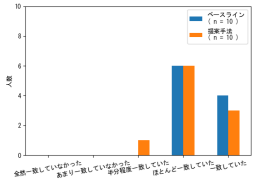


図 12: グラフ全体が記憶と一致すると思うか

6 考　　察

6.1 RQ1 についての考察

5.3.1 項，5.3.3 項で得られた結果から，参加者の印象としては，ほとんどの人が過不足なく閲覧履歴が記録されていると感じていることが分かった．

一方で，5.2.1 項で得られた結果では，課題 1 のベースライングループの正答率が 100%であるが，5.2.3 項で得られた結果では課題 1 のベースライングループの正解ノードの合計閲覧時間の最小値は 0 となっている．つまり，ベースライングループが全員正解しているにも関わらず，正解ノードを閲覧していない参加者がいるということになっている．

これらのことから，ユーザの印象レベルでは正確に閲覧履歴が記録・可視化がされていると感じているが，データレベルでは閲覧履歴に誤差が発生していることが分かる．今回の実験では短期的な使用による評価であったため，得られた閲覧履歴のレコード数が少なかった．誤って記録された閲覧履歴レコードが比率として多くなったが，本来の目的に則した長期的な使用であれば記録間違いは全体の割合としては少なくなると考えられる．そのため，今回のような短期的な実験では，ある程度の誤差はあるもののユーザ自身による評価では閲覧履歴は正確に記録されていると感じることが明らかになったが，長期的な使用において，どの程度の誤差がソースコードの理解に対してマイナスな影響を及ぼすかについても調査する必要がある．

6.2 RQ2，RQ3 についての考察

5.2.1 項で得られた結果から，提案手法グループはベースライングループに比べて課題の正答率が下がっていることが明らかになった．また，提案システムによる可視化が正答に関連する情報を提供できたと仮定するならば，ベースライングループよりも提案手法グループが早く正解ノードを発見することができ，長く正解ノードを閲覧すると考えられるが，5.2.3 項で得られた結果からは，両グループ間の正解ノードの閲覧時間に有意差は見られなかった．一方で，5.3.2 項で得られた結果から，課題の回答に他者の閲覧履歴グラフが役に立ったかという提案手法グループに対するアンケートの設問では，どの課題でも参加者の半数以上がとても役に立った・少し役に立ったと回答している．

このことから，参加者は提案システムによる閲覧履歴の可視化結果をプログラム理解の補助として利用することは出来ていたが，実験課題に正答できるほどの効果は得られなかったと考えられる．その原因として，提案システムによる閲覧履歴グラフの読解には，コードリーディングと類似した能力が必要であると推察される．閲覧履歴グラフでは，ネットワークグラフ上にどんなクラスやメソッドを示すノードが存在し，エッジを通してどこに関連性が存在しているかを発見することは可能である．しかし，その関連性にどのような意味があるのかまでを察することはコードリーディングの能力が高いユーザでなければ難しい可能性が考えられる．

今回の 4 つの実験課題は 2 種類のプログラム理解の能力が必要となるように設定している．1 つは,「正答に必要なクラスやメソッドを適切に発見することが出来るか」という能力を測る狙いのもと，機能追加を行う課題を設定した．課題 1 と 3 がこれにあたり，これらの課題では，既存のソースコード中の 1 行をほとんどそのままの形で適切な箇所にコピーすれば正答できるようになっている．もう 1 つは,「クラスやメソッド同士の関連性の意味を解釈することが出来るか」という能力を測る狙いのもとバグ修正を行う課題を設定した．課題 2 と 4 がこれにあたり，これらの課題では，親クラスや派生クラスのソースコードから，バグの原因を理解すれば正答できるようになっている．その上で，5.2.1 項で得られた結果では，課題 1 と 3，課題 2 と 4 で比較すると，両手法とも課題 2 と 4 の正答率が顕著に低くなっている．このことから，今回の実験の参加者自身が持つコードリーディング能力について，単純なクラスやメソッドの発見は可能であるが，その関係性の意味を適切に解釈することが難しく，本提案システムによる閲覧履歴グラフからソースコード同士のつながりを理解し，正答に至った参加者は少数だったのではないかと推察される．

したがって，提案システムによる閲覧履歴の可視化は単純なクラスやメソッドの存在確認に対しては一定の効果が得られるが，発見したクラスやメソッドから全体の関連性や構造を解釈し，理解するにはユーザ自身の読解力が必要であると考えられる．

7 結　　論

本研究における評価実験では，多少の誤差はあるものの，ユーザの印象では正しく記録・可視化がされていると感じる程度の閲覧履歴の精度があることが分かった．このことから，本研究において記録された閲覧履歴はある程度ユーザの閲覧行動を網羅することが可能であり，この閲覧履歴をプログラム理解支援に活用できる可能性を示した (RQ1).

本研究では，ネットワークグラフという形で閲覧履歴の可視化を行ったが，クラスやメソッドの単純な存在確認には，一定の効果があることが示唆された (RQ2).

しかしながら，グラフを提示するだけでは，提案システムの対象ユーザとなるプログラム理解の促進が必要な開発者にはソースコード全体の構造や関連性の理解を促進する効果は確認出来なかった．閲覧履歴グラフにはコードリーディング能力に類似した高い読解力が必要であることが考えられる (RQ3).

また，本研究では，提案システムのプログラム理解の促進について，利用しない場合と比べて顕著な効果は見られなかったが，閲覧履歴自体からは様々な分析が出来る可能性があることが実験から明らかになった．今後の研究では，記録・可視化両方の機能を改善していくとともに，プログラム理解支援の文脈だけでないプログラムに関する情報行動を分析するシステムとして本提案システムの別の活用方法も模索していくことを検討していきたい．

文　　献

[1] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. An examination of software engineering work practices. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '97, p. 21. IBM Press, 1997.

[2] M.P. Robillard, W. Coelho, and G.C. Murphy. How effective developers investigate source code: an exploratory study. *IEEE Transactions on Software Engineering*, Vol. 30, No. 12, pp. 889–903, Dec 2004.

[3] Katsuhisa Maruyama, Takayuki Omori, and Shinpei Hayashi. A visualization tool recording historical data of program comprehension tasks. In *Proceedings of the 22nd International Conference on Program Comprehension*, ICPC 2014, p. 207–211, New York, NY, USA, 2014. Association for Computing Machinery.

[4] 鈴木宏紀, 山本晋一郎, 阿草清滋. プログラム実行情報の視覚化による理解支援ツール. 情報処理学会研究報告ソフトウェア工学（SE）, Vol. 1998, No. 64(1998-SE-120), pp. 77–84, July 1998.

[5] Sourcetrail - the open-source cross-platform source explorer. `https://www.sourcetrail.com/`. (Accessed on 05/14/2021).

[6] Understand - ソースコード構造解析ツール. `http://understand.techmatrix.jp/`. (Accessed on 05/20/2021).

[7] Y. Cheng, W. Hsiung, Y. Wu, and L. Chen. Geekynote: A technical documentation tool with coverage, backtracking, traces, and couplings. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 73–76, Los Alamitos, CA, USA, oct 2020. IEEE Computer Society.

[8] C. Oezbek and L. Prechelt. Jtourbus: Simplifying program understanding by documentation that provides tours through the source code. In *2007 IEEE International Conference on Software Maintenance*, Los Alamitos, CA, USA, oct 2007. IEEE Computer Society.

[9] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. How do professional developers comprehend software? In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, p. 255–265. IEEE Press, 2012.

[10] Andrew Forward and Timothy C. Lethbridge. The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering*, DocEng '02, p. 26–33, New York, NY, USA, 2002. Association for Computing Machinery.

[11] J. Kotula. Source code documentation: An engineering deliverable. In *Technology of Object-Oriented Languages, International Conference on*, Vol. 1, p. 505, Los Alamitos, CA, USA, aug 2000. IEEE Computer Society.

[12] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. Eye movements in code reading: Relaxing the linear order. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, ICPC '15, p. 255–265. IEEE Press, 2015.

[13] Renske Talsma, Erik Barendsen, and Sjaak Smetsers. Analyzing the influence of block highlighting on beginning programmers' reading behavior using eye tracking. In *Proceedings of the 9th Computer Science Education Research Conference*, CSERC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[14] Eduarda Mendes Rodrigues, Natasa Milic-Frayling, Marc Smith, Ben Shneiderman, and Derek Hansen. Group-in-a-box layout for multi-faceted analysis of communities. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pp. 354–361, 2011.

[15] Visual studio code – コード エディター — microsoft azure. `https://azure.microsoft.com/ja-jp/products/visual-studio-code/`. (Accessed on 05/20/2021).

[16] Stack overflow developer survey 2019. `https://insights.stackoverflow.com/survey/2019`. (Accessed on 05/20/2021).

[17] Extension api — visual studio code extension api. `https://code.visualstudio.com/api`. (Accessed on 01/10/2022).

[18] D3.js - data-driven documents. `https://d3js.org/`. (Accessed on 12/24/2021).

[19] microsoft/vscode-vsce: Vs code extension manager. `https://github.com/microsoft/vscode-vsce`. (Accessed on 12/24/2021).

[20] nanaka0012/crhv-stg. `https://github.com/nanaka0012/crhv-stg`. (Accessed on 12/25/2021).

[21] gosu/gosu-examples: Example game collection for ruby/gosu, add yours by sending a pr! `https://github.com/gosu/gosu-examples`. (Accessed on 12/25/2021).