Vol.2013-SE-182 No.18
2013/10/24

プログラム理解支援を目的とした分散ペアプログラミングのコミュニケーションログの活用

秀毛 嶺維馬[1,a)]　奥野 拓[2]

概要：ソフトウェア開発において，他者が書いたソースコードを読解しなければならない状況が発生する．特に，コードレビューやシステムの機能拡張においては，プログラムを詳細に理解することが必要である．一般的には，コード中のコメントやドキュメント，実装の担当者の説明をプログラム理解の手がかりにする．しかしこれらの手がかりが無い場合，プログラム理解には非常に時間が掛かる．そこで本研究では，分散ペアプログラミングにおけるコミュニケーションに注目し，プログラム理解の効率化を図る手法を提案する．特に本稿では手書き注釈によるコミュニケーションログの活用に焦点をあて，その有効性について検証した結果を述べる．

1. はじめに

離れた二者が共にコーディングする分散ペアプログラミングという手法がある [1]．この手法では，作業者の間で交わされるコミュニケーションの記録が容易である．通常の分散ペアプログラミングでは作業者らのコミュニケーションは意図伝達以上に活用されることはない．しかし，コーディング時のコミュニケーションには作業者らの意思決定が含まれているため，何らかの活用が考えられる．

ソフトウェア開発において，他者が書いたソースコードを読解しなければならない状況が発生する．特に，コードレビューやシステムの機能拡張の際には，プログラムの細かな理解が必要である．一般的には，ソースコード中のコメントやドキュメント，実装の担当者の説明をプログラム理解に活用する．しかしこれらの手がかりが無い場合，プログラムの理解には非常に時間が掛かる．

本稿ではこの問題に対して，分散ペアプログラミングのコミュニケーションログを用いてプログラムの理解支援を行う．特に手書きの注釈によるコミュニケーションに焦点をあて，本稿ではその有効性について述べていく．

2. 分散開発におけるプログラム理解支援

2.1 プログラム理解支援の必要性

ソフトウェアの開発において，開発メンバが分散するケースがある．有志によるオープンソースソフトウェアの開発や，海外や国内に開発の一部を委ねるオフショア開発・ニアショア開発はその一つの例である．この分散開発では，一つの場所に集合して開発する場合に比べ，密なコミュニケーションが取りにくい．コミュニケーションはグループウェアを介して行う必要があり，直接対話するよりも手間が掛かる．そのため，分散開発ではコミュニケーションロスが発生しやすい状況にあると言える．

特にプログラムの実装においては，密に連携できないことが作業効率に影響する．例えば他人が実装したソースコードをリファクタリングする際，実装されたコードに疑問が生じても，開発担当者への照会に時間が掛かる．また，開発ドキュメントのメンテナンス不足や内容が十分ではない場合，ソースコードの保守は困難になる．したがって，作業者個人で利用可能なソースコード理解手法が必要だと考えられる．

2.2 既存のプログラム理解の手法

一般に，クラスやメソッドの仕様を理解するには，開発ドキュメントの読解を行う．また，デバッガを利用して変数の状態を監視する，テストケースから仕様を推測するなどといった方法もある．ソフトウェアの保守作業においては，そのソースコードがどのように変更されてきたかを知る必要がある．

そこで版管理システムを利用したソフトウェア開発では，ソースコードの変遷を知るためにコミットログを参照する．コミットログにはソースコードの差分の他，ソースコードの変更理由などをコミットメッセージとして残し

[1] 公立はこだて未来大学大学院
Graduate School of Future University Hakodate
[2] 公立はこだて未来大学
Future University Hakodate
[a)] g2112020@fun.ac.jp

ておける．しかし，このコミットメッセージの有無は作業者に依存するため，必ずしも付けられるものではない．また，その内容が適切に更新内容を説明しているとも限らない．さらに，詳細なソースコードの変化を追うためには，コミット単位の調査では不十分である．

版間で同じソースコードを何度も編集されていても，その試行錯誤は記録には残らない．その試行錯誤の経過を辿ることで開発者の意図を探ることが可能と考えられるが，コミット単位の記録ではそのような調査は行えない．そこで，統合開発環境への入力を記録した編集操作履歴を利用することが試みられている [2]．編集操作履歴を利用することで，ソースコードの変化をより詳細に追うことができる．しかし，この編集操作履歴にも幾つかの課題が存在する．

3. 編集操作履歴によるソースコード変遷追跡における課題

3.1 編集操作履歴の量

プログラムの統合開発環境では，ファイル内容の変更差分が編集操作履歴として残される．この編集操作履歴から，ソースコードの変遷を追うことができる．しかし，ステップ数の多いソースコードや頻繁に編集されたソースコードは編集操作履歴の量が膨大となるため，ソースコードの特定箇所を参照するのに時間がかかる．

3.2 編集理由の類推

編集操作履歴には，コミットメッセージのように何を意図して変更されたかは記述されていない．版管理システムではコミットメッセージを残すことができるが，ソースコード一行単位でどのような意図をもって変更されたかは類推するしかない．ソースコードの細かい部分の変更意図を知りたい場合，編集操作履歴やコミットメッセージでは不十分である．

4. プログラム理解支援の関連研究

編集操作履歴の調査時間を短縮する手法として，大森らが提案した編集操作再生器がある [3]．これは，編集操作履歴から任意の時点のソースコードエディタを再現することができる．さらに，編集操作履歴の内容を時系列に並べ、利用者に図示する．大森らの行った評価実験では，提案手法を用いない場合 8 分掛かっていた調査時間が，1 分に短縮されたという結果が得られている．このタイムラインによる編集操作履歴の視覚化は，編集操作履歴の調査時間の短縮に貢献するものだと考えられる．

次に，編集理由の類推を支援する手法として，西川らの提案したコミュニケーションログの利用がある [4]．西川らは分散ペアプログラミング中に交わされるコミュニケーションが編集理由の類推に有用と考え，テキストチャットと編集操作履歴を紐付けて提示するという手法を提案した．この手法の評価実験では，編集操作履歴の 68.8 %がテキストチャットのログとほぼ対応すると評価された．しかし，テキストチャットのログには，相槌やコーディングとは無関係な対話など不必要な情報が含まれる．これらは，目的の情報を見付けるためのノイズとなる．そのため、これらのノイズを取り除くような工夫が必要である。

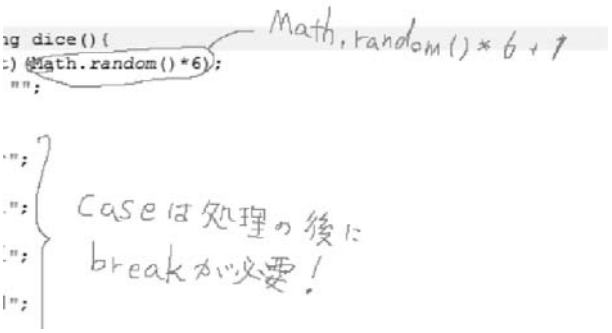

図 1 コードに付記された手書き注釈

表 1 対話手法毎の利用状況

対話手法	利用状況	頻度
テキストチャット	Web サイトの文章の引用	低
音声通話	ソースコードの説明 疑問点への問いかけや対応	高
手書き注釈	図表による音声通話の補助 記号による注目点の指示 気づいたことのメモ	中

5. 手書き注釈ログを利用したプログラム理解支援

本稿では 3 節で述べた編集操作履歴の読解に関する問題解決に取り組む．以下に既存の支援手法と問題解決のための提案を述べる．

5.1 手書き注釈ログの利用

本研究では編集理由の類推の簡易化という課題に対して，手書き注釈によるコミュニケーションログに注目した．ここでの手書き注釈とは，図 1 のようにソースコード上に付記された文字や下線，記号を指す．表 1 は分散ペアプログラミングにおいて，テキストチャット，音声通話，手書き注釈がそれぞれどのような状況で使われるかを示したものである [1]．利用の容易さや頻度から，相槌やコーディングと無関係な会話など，ノイズとなりうる情報が一番多く含まれているのは音声通話だと予測できる．一方で，手書き注釈は注目点の指示や気付きの伝達に利用されている．多くはソースコードの一部分に関する説明を補助することに使われている．そのため，手書き注釈により補助された音声通話は，プログラムの編集理由の類推に有用と考えられる．

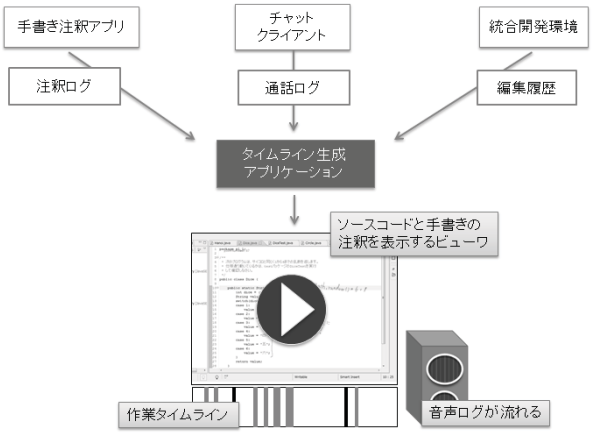

図 2 提案システムの全体図

表 2 検証 1 の試行結果

	所要時間	手書き注釈	注釈数
ペア 1	07m05s	あり	15
ペア 2	09m05s	なし	-

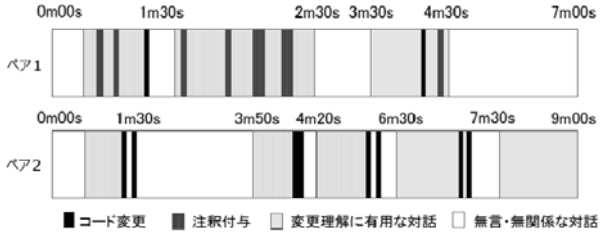

図 3 検証 1 における被験者の作業タイムライン

5.2 編集操作履歴と手書き注釈ログの可視化

5.1 節から，手書き注釈が付けられた付近の音声通話を抽出することで，プログラム理解に有用な対話をユーザに提示できると考えられる．そこで，編集操作履歴とコミュニケーションログを収集し，ユーザにそのログを提示するシステムを提案する．システムの全体図を図 2 に示す．

まずソフトウェア開発フェーズにおいて，分散ペアプログラミングに利用したツールから各種ログを収集する．具体的には，統合開発環境からは編集操作履歴，手書き注釈アプリケーションからは手書き注釈ログ，チャットクライアントからは通話ログを収集する．そしてソフトウェア保守フェーズで，保守担当者がログをビューワから視聴する．このビューワは大森らの編集操作再生機と同様，編集操作履歴をもとに，任意の時点のソースコードエディタを再現する．エディタの下部の作業タイムラインからは，コードの編集と手書き注釈が付記された時刻を把握できる．また，動画のように任意の時点から音声通話記録を再生できる．保守担当者はエディタ下部のグラフを頼りに目的の編集箇所を探索する．そして，手書き注釈が付記されている付近の音声通話記録を視聴し，これをプログラム理解の手がかりとする．

6. 手書き注釈の有効性検証

5 節で述べた，手書き注釈を利用したプログラム理解支援手法の有効性を測るために，2 つの検証を行った．以下にその方法と結果について述べる．

6.1 検証 1：注釈の前後の対話に関する検証

5.1 節から，手書き注釈が付けられた付近に，ソースコード変更理由の類推に有用な対話があると仮説を立てた．この仮説を検証するため，実験を行った．また，注釈の有無によるログの変化も考察した．

6.1.1 実験方法

被験者には講義で Java プログラミングの基礎を履修した，情報系大学の学部生 4 名を選んだ．この 4 名で二組のペアを作り，分散ペアプログラミングで Java で書かれたプログラムのデバッグを行わせた．

課題にはサイコロのように 1 から 6 をランダムに出力する，20 ステップ程のソースコードを用意した．このソースコードにはシンタックスエラーは無いが，仕様とは異なる 3 つのバグが存在している．デバッグを行うクラスファイルの他にテストケースを用意し，このテストを通過するまでを試行とした．

被験者らは二組ともコーディングには統合開発環境を利用したが，コミュニケーションの手段に差を設けた．一方はテキストチャットと音声通話を利用した．もう一方はそれらに加え，手書き注釈によるコミュニケーションを行った．手書き注釈の付記には，ペンタブレットを利用した．

分散ペアプログラミングでは，作業者はソースコードを編集する実装役とアドバイスを行う補助役に役割が分かれる．通常，15 分から 30 分でこの役割は交代するが，今回の試行は所要時間が短かったため，役割の交代はしなかった．

6.1.2 実験結果

実験の様子は撮影し，動画を元にコード変更箇所と注釈付与の箇所，コード変更理由を類推するのに有用な対話と判断した箇所を図示した．手書き注釈がある場合とない場合で作業タイムラインの比較を行った．

試行の結果を表 2 に示す．ここでの注釈数は，ペンタブレットのペンが接地してから離れるまでを 1 回と数えている．また，これらの試行の動画から作成した図を図 3 に示す．

図 3 のペア 1 における記録から，手書き注釈が付記された前後でコード変更理由となる対話がされていたと読み取れる．今回の試行では，付記された注釈はソースコードの特定箇所を指すために利用されていた．被験者の発言内容を観察すると，コード変更前はバグの出所について話されていた．またソースコード変更後は，その時の変更がどのような物だったかソースコードに注釈を付記しながら話さ

表 3 検証 2 の試行結果

所要時間	被験者 1 注釈数	被験者 2 注釈数
03h21m25s	284	15

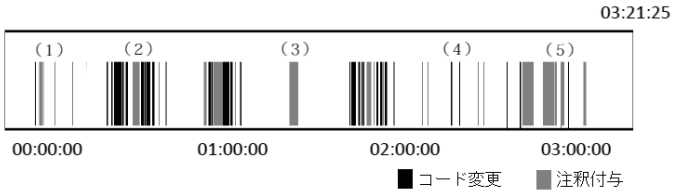

図 4 検証 2 における被験者の作業タイムライン

れていた．手書き注釈の無いペア 2 の試行からは，コード変更数が手書き注釈を利用した場合に比べ多いことが読み取れる．こちらもコード変更の前後でソースコード変更の手がかりとなる内容の対話がされていた．

6.1.3 考察

この検証で行われた試行では，仮説の通り，手書き注釈の付記された前後でコード変更理由となる対話がされていた．手書き注釈が無い場合は，コード変更理由となる対話は実際にコード変更された前後でされていた．しかし，コード変更の回数が手書き注釈有りの場合に比べ増加した．これは，言葉で伝えるより先に一時的にソースコードを変更して，結果を協調作業者に見せていたためである．手書き注釈が無い場合でもコードが変更された前後の対話を参照すれば，コード変更理由の類推は可能であると考えられる．しかし，一時的な変更によってコード変更箇所が増加し，変更履歴が冗長になった．後にプログラム理解をするにあたって，変更履歴は少ないほど調査する時間が減少する．そのため両者を比較した場合，手書き注釈がある方が優位と考えられる．

6.2 検証 2：長時間の試行での検証

6.1 節で行った検証は，試行時間が 10 分以内と短いものだった．そのため交わされた対話が少なく，提案手法が有効な条件を確認するには至らなかった．そこで課題の難易度を上げ，再度実験・考察を行った．

6.2.1 実験方法

被験者には情報系の大学院生 2 名を選んだ．被験者 1 は講義以外でも Java アプリケーションの実装経験があるが，被験者 2 は講義以外では Java はほぼ扱っていなかった．この二人でペアを組み，分散ペアプログラミングを行った．課題は Java の連想配列，コレクションの操作に関するものを用意した．この課題はプログラミングの講義で利用されたものであり，3 時間程度の時間で解くことを想定して作成されている．被験者にはプログラムの仕様と参考資料，一部のメソッドが未実装のクラスファイル，テストケースを提供した．

6.1 節の実験と同じく，実装には統合開発環境を利用した．また，コミュニケーションの手段としては音声通話，テキストチャット，手書き注釈を利用した．先の実験とは違い，実装役・補助役の交代は 20 分毎に行った．

6.2.2 実験結果

実験の様子を撮影し，動画を元にソースコードの編集・手書き注釈が付記された箇所を図示した．

試行の結果を表 3 に示す．また，試行中のソースコードの編集・手書き注釈の付記を可視化したものを図 4 に示す．図 4 の特徴的な箇所における作業の状況について，以下に述べる．

（1）では，被験者らはプログラムの仕様や実装するメソッドの確認を行っていた．(2）では，作業方針が固まり，順調にメソッドを実装していた．途中，記述の間違いを指摘し，指示するのに注釈が用いられた．(3）では，それまでの実装に区切りがつき，次に実装する内容について処理の流れを確認しあっていた．変数に格納されるデータの状態について，表を描いて説明していた．(4）では，実装したメソッドのテストが通らず，配布資料の再読や API の調査をしながら試行錯誤していた．ここでは，注釈による補助は行われなかった．(5）では，最後のテストを通すため，処理の流れについて再検討していた．説明には図が用いられた．

試行中はお互いにソースコードを編集する直前や，その最中に自分の考えを発話していた．しかし，Java の実装経験のある被験者 1 に比べ，講義で基礎を学んだのみの被験者 2 は注釈付記数は少なかった．

今回の試行で記述された注釈を図 5，図 6 に示す．また，注釈の形状と対話内容について，以下に記述する．矢印・丸囲い・下線は，主に補助役からの指摘，注目点を伝える際に利用されていた．この指摘の際には文字を伴うことがあり，これは描画した文字を入力するよう指示することに使われていた．図や表が描かれた際には，ある変数に格納されたデータの説明や，関数のワークフローについて説明がされていた．

6.2.3 考察

実験結果より，手書きの注釈が付記されにくい状況があることが分かった．一つは被験者らのコミュニケーションに問題がある場合，もう一つは被験者らが独立して作業している場合である．以下に詳細を述べる．

まず表 3 より，被験者 1 に比べ，被験者 2 による注釈付記数が少ないことが分かる．また，その内容が図 5 のような指摘に用いる注釈だったことから，被験者 1 による一方的なコミュニケーションが記録の多くを占めていたと言える．

被験者 2 の注釈数が少ないのは，ペアの組み合わせに起因すると考えられる．ペアプログラミングにおいて，実装

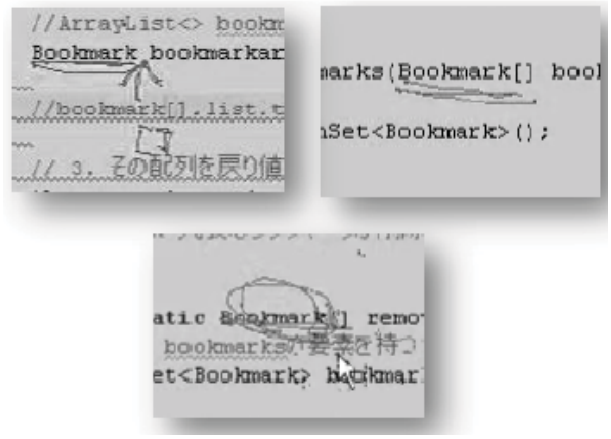

図 5 注目点の指示に用いられた手書き注釈：矢印・下線・丸囲い

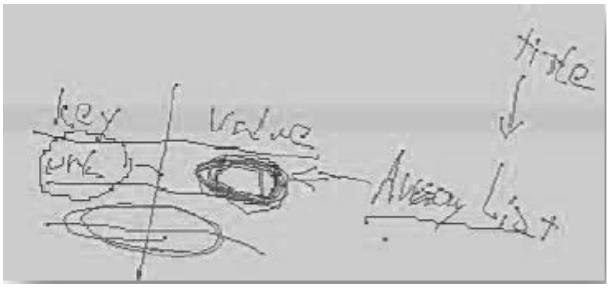

図 6 変数の内部状態を示すのに用いられた手書き注釈：図・表

の熟練度に差があるプログラマの組み合わせでは，発言数が少なくなる状況がある．これは熟練度が高いプログラマが実装役，低いプログラマが補助役に就いた際に起こる [5]．熟練度の低いプログラマは補助役として指摘できる箇所が見当たらず，コードの記述に疑問が生じても質問しない．熟練度の高いプログラマは補助役からの質問がないことを，相手が自分のやろうとしていることを理解していると考え，そのまま作業を進めてしまう．対話数が減少することで，ペアプログラミングの利点である知識共有が行われない．また，注釈の数も対話数に伴って減少する．

この組み合わせにおいて発言機会を減少させないためには，次の二点がポイントになる．一つは，熟練度の高いプログラマは相手の理解度の確認する機会を設ける，質問できる環境を作ることである．もう一つは，熟練度の低いプログラマは積極的に質問することである．

今回の試行では，実装役と補助役を定め，定期的に交代する以外は作業の進め方に制限を設けていなかった．今後の実験では発言機会の減少を防ぐため，作業者らの属性に応じた，対話の仕方を事前に教示する必要があると考えられる．

また図 4 の（4）では，注釈による補助が行われなかった．ここでは，作業者らは実装に試行錯誤しており，それぞれ別の作業を行っていた．例えば，実装役がソースコードの記述を試行錯誤している間に，補助役が参考資料の調査をしていた．このように，各々の意見がまとまっていない場合には，対話が少なく，注釈による意図伝達も行われないと考えられる．

7. おわりに

本稿では，分散ペアプログラミング中に蓄積されるコミュニケーションのログを利用して，プログラム理解を支援する手法を提案した．従来手法ではコミュニケーションログのノイズが課題となっていた．そこで，ソースコードエディタ上に付記された手書きの注釈が，音声通話の補助に使われていたことから，注釈が描かれた付近の音声通話ログを抽出することで，ノイズを軽減できると仮説を立てた．

仮説を検証するため，二つの実験を行った．検証では，仮説が成り立つ状況があることを確認した．しかし，被験者らのコミュニケーションが少ない場合や，独立して作業している場合には，注釈が付記されないことがわかった．

今回は検証例が少ないため，更なる試行が必要である．その際には，被験者等のコミュニケーションの内容の分析のため，作業中におけるコミュニケーションのガイドラインを提示することを検討する．それに併せて，ソフトウェアの保守作業時を想定した検証を行っていく．

参考文献

[1] 秀毛嶺維馬，奥野拓：分散ペアプログラミングにおける手書き注釈を用いたコラボレーション機能の提案，情報処理学会北海道シンポジウム講演論文集, pp.175-180, (2012)
[2] 大森隆行，丸山勝久：開発者による編集操作に基づくソースコード変更抽出，情報処理学会論文誌，Vol.49，No.7，pp.2349-2359，(2008)
[3] 大森隆行，丸山勝久：プログラム開発履歴調査のための編集操作再生器，ソフトウェア工学の基礎 XVII, pp.45-54, (2010)
[4] 西川穂高，酒井三四郎：分散ペアプログラミングにおけるコミュニケーションログとコード変更箇所の対応付けによる理解支援, 情報科学技術フォーラム一般講演論文集, pp.103-104, (2005)
[5] Laurie Wiliams and Robert Kessler：Pair Programming Illuminated, Addison-Wesley Professional (2002)