修士論文概要

細粒度履歴を用いたプログラミング過程の可視化ツール†

髙山　源貴*

（2019年３月８日受理）

A Visualization Tool for Programing Process Utilizing Fine-Grained Development Records

Genki Takayama

（Received March 8, 2019）

１　はじめに

ソフトウェア開発においてソースコードを管理する際にバージョン管理システムが用いられる。バージョン管理システムとは、ソースコードの変更履歴を記録しておくものであり、過去のソースコードの確認やソースコードの復元などに用いられる。

しかし、バージョン管理システムへの記録は、開発者本人のタイミングで行われるため、必要な記録が残っているとは限らない。

そこで本研究では、ソースコードの全ての変更過程を細粒度履歴として自動的に記録し任意の時点を閲覧することが可能なツールの開発を行った。

２　研究目的

本研究では、統合開発環境（IDE）上で細粒度履歴の記録を行う環境を構築し、web上で記録の再現・可視化を行うアプリケーションの開発を行う。

また、ツールの活用方法について以下の２つの提案・実験を行う。

１．プログラミング過程の理解

２．プログラミング傾向の分析

１つ目のプログラミング過程の理解とは、他者の作成したソースコードのプログラミング過程を見ることでプログラミングの意図を理解し原作者が行うと考えられる変更を予想たソースコードの変更を行うことによりソースコードの一貫性を保つことができる。

２つ目のプログラミング傾向の分析とは、ソースコードの変更過程を分析することで、作成者のプログラミング手法、行き詰まり箇所、得意な箇所を見つけることができる。

３　提案手法

細粒度履歴の記録・再現・可視化を行うための機能として、「細粒度履歴の記録」、「細粒度履歴の再現」、「メトリクスの可視化」、「実行結果の表示」の開発を行った。

開発したツールの全体の構成は図１のようになっている。

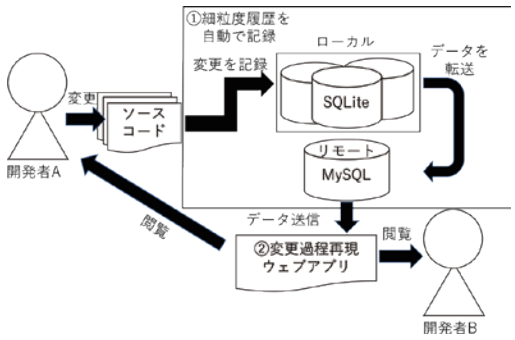

図１．ツールの全体の動き

利用者がソースコードを編集すると細粒度履歴の記録をするために編集されたデータを取得する。取得したデータはローカルのデータベースに記録される。次に、ローカルのデータベースからデータをリモートのデータベースへ転送しwebアプリケーション上で記録の再現やメトリクスの可視化、実行結果の表示を行う。

４　実装

開発したwebアプリケーションを図２に示す。

webアプリケーションは、６つの画面で構成されている。

①プロジェクトの選択、②ファイルの選択、③ソースコードの表示、④ソースコードの増減のグラフ、⑤ログの表示、⑥サイクロマチック数[1)]の表示。

①のプロジェクトの選択では、データベースに記録されているプロジェクトから表示したいプロジェクトの選択を行う。

②のファイルの選択では、選択したプロジェクトのファイルが表示される。ファイルは、プロジェクト名フォル

† 本研究の一部は、2019年３月に電子情報通信学会知能ソフトウェア工学研究会（KBSE）において発表。

* 電子情報メディア工学専攻2178010 高瀬研究室

ダ、パッケージ名フォルダ、ファイルの順番にツリー表示される。

③のソースコードの表示では、選択されているファイルのソースコードが表示される。ソースコード表示画面の上側にはコード、下側にはスライダが表示されている。スライダを操作することでコードがスライダで選択した時間の状態を表示する。また、スライダの左には、スライダが現在選択している時間が表示される。

④のソースコードの増減のグラフでは、選択されているファイルのソースコードの増減の変化が表示される。グラフは、縦軸がコードの文字数、横軸が入力した時間。 上側のグラフをクリックするとコードの表示画面がグラフをクリックした位置の時間のコードが表示される。

⑤のログの表示では、選択されているファイルのプロジェクト全体のログを表示する。ログの表示は、１行目にrun、debug、start、saveが表示される。runは実行、debugはデバック実行、startはコンソールの起動、saveはファイルの保存となっている。２行目以降には、実行またはデバック実行したときの出力結果、または実行時エラーが表示される。

⑥のサイクロマチック数の表示では、現在選択しているファイルのサイクロマチック数が表示される。表示は、メソッドが作成されたまたは、サイクロマチック数が変更された時間、メソッド名、サイクロマチック数が表示される。

図２．作成したwebアプリケーション

５　実験１プログラミング過程の理解

細粒度履歴を閲覧した被験者が、ソースコードを修正するにあたり、細粒度履歴を閲覧できない被験者と比べてプログラミングの意図を推測できているのか確認するために実験を行った。

実験は、日本工業大学情報工学科の学部３年生６名を対象に実施した。被験者には、Java言語で書かれたソースコードと、ソースコードを改良した後の出力結果が問題として与えられる。被験者は与えられた出力結果を手がかりとして、与えられたソースコードに変更を加え、与えられた出力結果と同一の出力をするようになった時、もしくは被験者がギブアップしたときに問題は終了する。実験は、IDE上で実装した機能が利用できる群とそうでない群に分けて実施し、機能の利用の有無が及ぼす影響を調べた。

５.１　実験１結果

実験の結果プラグインを使用した被験者は全員完成することができたのに対して、プラグインを使用しなかった被験者は１人だけ完成させることができなかった。

また、意図の推測については、プラグインを使用しなかった場合、あらかじめ予想していた変更内容と違う変更を行っていたのに対して、プラグインを使った場合は、2人が予想していた変更内容と同じ変更を行っていたため、この２人は意図の推測ができていたと考えられる(表１)。

表１．実験１の意図の推測

被験者	プラグインの使用	変更内容	意図の推測
A	無	returnの追加と出力結果の変更	×
B		出力位置の変更	×
C		未完成	×
D	有	初期値の変更	○
E		出力位置の変更	×
F		初期値の変更	○

５.２　実験１考察

結果から、プログラミングの意図を推測できていた被験者が、ソースコードを修正するにあたり、意図を推測できていない被験者と比べてどのような違いが出るのかについて考察を行う。

図３に６名の被験者Ａ～Ｆの実験中のソースコードの増減を示す。グラフ中の濃いグレーの網掛け部分は、問題を解くことに直接貢献するメソッドに対して変更を加えていたことを表している。また、上段のＡ～Ｃの被験者は細粒度履歴を閲覧できない群であり、下段のＤ～Ｆの被験者は細粒度履歴を閲覧できる群である。このうち、Ｄ、Ｆの２名の被験者（図３点線部分）については意図を推測できている被験者である。

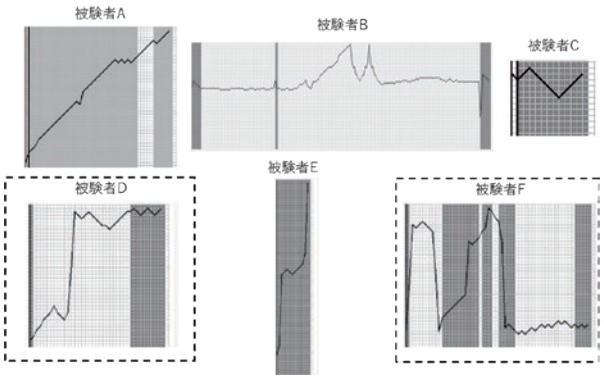

図３．実験１の各被験者のソースコードの増減

図３より、意図を推測できている場合とそうでない場合で、被験者のプログラミング過程に違いがあることが分かる。意図が推測できていない被験者はソースコード中のある１つのメソッドを集中的に編集する傾向があるのに対し、意図が推測できている被験者は問題を解くことに直接貢献するメソッド以外に対しても修正を加えながらプログラミングを行っている。よって、ソースコードの特定の場所を集中して編集している被験者はソースコードの全体の

構成に目が行き届かず、結果として原作者の意図よりも、開発者自身の意図が優先されてしまうのではないかと考えられる。一方で、ソースコード全体に満遍なく変更を行っている被験者は、ソースコード全体に目を通し、場合によっては問題を解くことに直接貢献するメソッド以外にも変更を加え、その挙動を確認していることが原作者の意図の理解に繋がっているのではないかと考えられる。

６　実験２プログラミング傾向の分析

細粒度履歴の再現・可視化機能を用いてプログラミング過程から各被験者にどのような特徴があるか分析するための実験を行った。

実験は、日本工業大学情報工学科の学部４年生６名を対象に実施した。被験者には、問題が２問与えられそれぞれの問題を解いてもらう制限時間は２問で１時間。問題は、１問目が完成またはギブアップしたら２問目を開始、２問目が完成またはギブアップで終了する。ソースコードは、問題で指定された結果となるソースコードをJavaで作成する。実験中のプログラミング過程は、常に記録しておき各被験者がどのようにプログラミングを行ったのか分析を行った。

６.１　実験２結果

表２と表３に実験の結果を示す。

表２．実験２の１問目の結果

被験者	所要時間	変更回数	1分当たりの変更回数	終了時の複雑度	完成
A	約14分	742回	52回	6	○
B	約25分	375回	15回	4	○
C	約45分	720回	16回	7	×
D	約59分	1366回	23回	0	×
E	約23分	1178回	51回	5	○
F	約27分	666回	24回	8	○

表３．実験２の２問目の結果

被験者	所要時間	変更回数	1分当たりの変更回数	終了時の複雑度	完成
A	約40分	2241回	56回	6	○
B	約34分	528回	16回	4	×
C	約13分	335回	25回	1	×
D	0分	0回	0回	0	×
E	約22分	851回	39回	4	×
F	約28分	467回	17回	4	×

実験の結果、２問とも完成した被験者が１人、１問目のみ完成した被験者が３人、１問も完成しなかった被験者が２人という結果になった。

また、１問目の結果では１分間当たりの変更回数が50回を超えている２人がほかの被験者よりも早く完成させていることから、作業量と完成までの所要時間は関係がないのだと考えられる。

次に、１問目のみ完成した被験者の１問目と２問目の変更回数を比較すると、被験者Ｂ以外の２人は完成しなかった場合の１分当たりの変更回数が減っていることがわかる。また、１分当たりの変更回数が増えている被験者Ｂもほぼ１問目と同じくらいのため、問題がわからなかった場合は分かった場合に対して１分当たりの変更回数が大きく増加することはないと考えられる。

６.２　実験２考察

実験中の各被験者のプログラミング過程を可視化しそれぞれのプログラミングの特徴について考察を行った。

その結果、プログラミング過程には以下の３つのパターンがあると考えられる（図４）。

１．前半の増加量が多いパターン
２．後半の増加量が多いパターン
３．増減が激しく変わるパターン

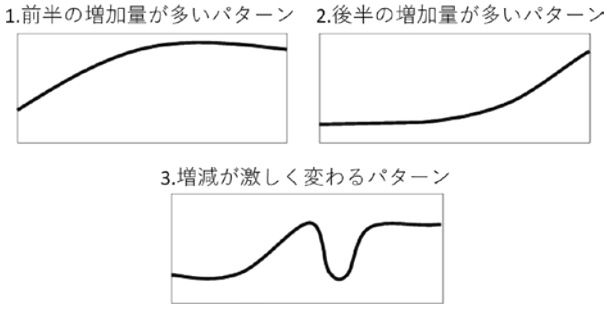

図４．プログラミング過程のパターン

まず、１のパターンは、問題１の被験者Ａと問題２の被験者Ｅが当てはまる。このパターンでは、時間に対しての変更量が多いという特徴がある。これは、プログラミングを始めてから早い時間でほぼ完成形に近づいているということであるため、完成形の予想がついた状態でプログラミングを始めて一気にその状態まで作り上げ最後に細かな修正を行っていると考えられる。また、ソースコードを完成できなかった被験者Ｅのデータから自分の思う結果にならなかった場合に再度初めから考え直すため進まなくなってしまうと考えられる（図５）。

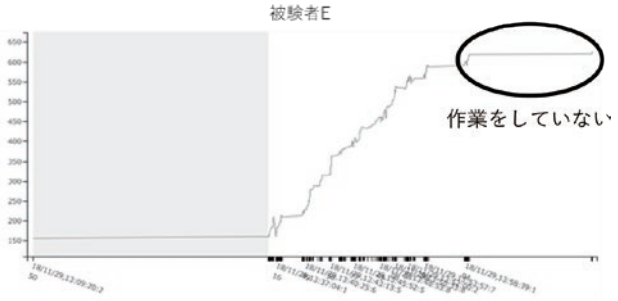

図５．被験者Ｅのソースコードの増減

次に、２のパターンは、問題１の被験者BCEFと問題２の被験者ACが当てはまる。

例として、被験者Ａの問題２のソースコードの増加量のグラフを図６に示す。このパターンでは、ソースコードを変更する際に１分前後の操作をしていない時間が細かく入るという特徴がある。これは、プログラミングの開始時には、完成形の予想がついていないため問題を読み解きながらアルゴリズムを考えている。そのため、ソースコードを書いてアルゴリズムを考えるというのを繰り返し行ってい

るため、このようなパターンになるのだと考えられる。

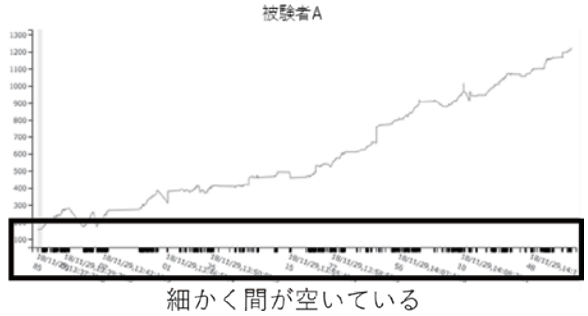

図６．被験者Ａのソースコードの増減

最後に、３のパターンは、問題１の被験者Ｄと問題２の被験者BFが当てはまる。

例として、被験者Ｅの問題１のソースコードの増加量のグラフを図７に示す。

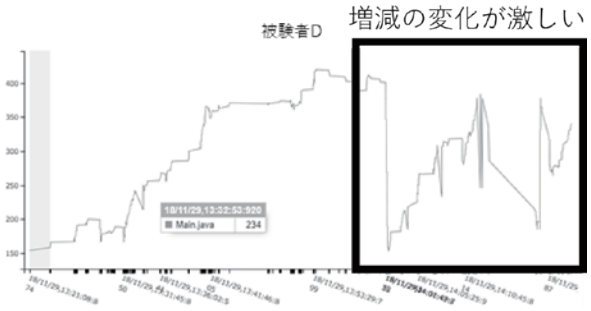

図７．被験者Ｄのプログラミング増減

このパターンでは、書いたコードの半分以上を一気に消すという特徴がある。これは、２の後半の増加量が多いパターンと同じように完成形がわからない状態からプログラミングを開始しているが後半の増加量が多いパターンが徐々に完成形に近づいていくのと違い、途中でこのままでは完成しないことを理解しやり直すという行動が表れているのだと考えられる。そのため、このパターンの場合は完成形がわからないまま進め続けているのではないかと考えられる。

また、今回収集したプログラミングのデータから、５分以上変更を行わず実行などの操作もしていないまたは、改行や空白の追加や削除などの変更のみを行っている場合が見て取れた。これは、１問目の被験者Ｃと２問目の被験者Ｅに見られた特徴である。被験者Ｃは、５分以上操作を行わなかった後にソースコードの動作に関係のない操作をしてから２問目に進んでいる。また、被験者Ｅは、作業が止まってから終了時間になるまで操作を行わなかった。

これらのことから、５分以上変更を行わず実行などの操作もしていないまたは、改行や空白の追加や削除などの変更のみを行っている場合その問題をわからないものとしてあきらめている可能性が高いと考えられる。

そして、増減が激しく変わるパターンは行き詰まり状態な可能性がある。このパターンになってしまう場合、プログラミング言語の使い方が理解できていないもしくは、問題のアルゴリズムがわからない可能性が高いと考えられる。

７　関連研究

大森ら[2)]の研究は、プログラミングの理解が重要でそのためにソースコードがどのように変更されたのかを知る必要があるという考えで、ソースコードの復元ツールの作成を行い実用性を持つことが示されている。また、藤原ら[3)]の研究は、演習中の学習者の特徴を自動的に判別して学習状況を判別するという研究で、学習者の時系列データをグラフ化しグラフの特徴から学習状況を判別している。

これらの研究では、ソースコードかグラフのどちらかしか重要視していないのに対して本研究では両方を組み合わせて分析を行うことができる。

８　終わりに

本研究では、ソースコードの作成過程に着目し、細粒度履歴の記録を行うIDEのプラグインと記録の再現とメトリクスの可視化を行うwebアプリケーションを開発した。

実験の結果から、意図を推測するためには、ソースコードの特定の場所を集中して編集するのではなく、問題を解くことに直接貢献するメソッド以外にも変更を加えるなど、ソースコードの全体に対して注意を払う必要があることが分かった。また、このツールを使い各被験者のソースコードの増減の量と各メソッドへの変更回数の違いを確認することができた。そして、プログラミング過程には３つのパターンがあることが分かった。

今後の展望として、プログラミングのパターンが実際に有効であるか検証を行う必要がある。

参　考　文　献

1）THOMAS J.McCabe, “A complexity measure,” IEEE Transactions on Software engineering, vol.SE-2, no.4, pp.308-320,1976.
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，pp. 2349-2359 (July 2008).
3）藤原理也，田口浩，島田幸廣，高田秀志，島川博光，“ストリームデータによる学習者のプログラミング状況把握,” DEWS2007 D9-5 (2007).

論文審査委員

審査委員（主査）　教　授　高瀬浩史
審査委員（副査）　准教授　松田　洋
審査委員（副査）　准教授　勝間田仁
助　教　橋浦弘明