様　式　Ｃ－１９、Ｆ－１９、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成 ２８ 年　 ５ 月 ３０ 日現在

機関番号：３４３１５
研究種目：基盤研究(C)（一般）
研究期間：2012～2015
課題番号：２４５０００５０

研究課題名（和文）プロアクティブ型プログラム変更支援環境

研究課題名（英文）An Environment that Proactively Supports Program Evolution

研究代表者
丸山　勝久（Maruyama, Katsuhisa）

立命館大学・情報理工学部・教授

研究者番号：３０３３００１２

交付決定額（研究期間全体）：（直接経費）　　4,200,000円

研究成果の概要（和文）：本研究では、開発者や保守者が過去に行ったプログラムのソースコードの編集操作の履歴から、将来のプログラム保守に役立つ情報を提示する2つの手法を提案した。一つは、過去のプログラム変更を開発者や保守者の方針に応じて自動的に検出・集約する手法である。もう一つは、連続する2つの編集操作間に存在する依存関係を形式的に表現する編集操作グラフを用いて、特定のクラスメンバの作成に関係する編集操作だけを抽出・再演する編集操作スライシング手法である。これらの手法を組みこんだツールをそれぞれ試作し、統合開発環境に組みこむことで、プログラム変更に対する有用性を示した。

研究成果の概要（英文）：In this research study, we improved a tool that automatically records fine-grained edit operations during the construction of a program and proposed two methods that can support future maintenance of the program. One of the methods implements automatic detection of program changes based on programmers' policies with respect to aggregation of recorded edit operations. The other method slices the edit operation history and replays its resulting slices, using a graph that represents the dependencies among edit operations in the history. We also developed powerful tools that implement the respective methods and demonstrated their usefulness for supporting program evolution.

研究分野：ソフトウェア工学

キーワード：ソフトウェア保守と進化　プログラム変更支援　プログラム理解　ソフトウェア開発環境

１．研究開始当初の背景

ソフトウェアが重要な役割を果たす現代社会において、信頼性や安全性の高いソフトウェアの構築がますます求められている。しかしながら、このようなソフトウェアを開発の当初から完璧に構築することは難しい。さらには、ソフトウェア進化の法則によると、ソフトウェアは、本質的に出荷後も利用者の要求の変化や外部環境（開発環境や実行環境）の変化に応じて、継続的に変更され続けなければならない。変化に追従できないソフトウェアの信頼性や安全性は相対的に低下するため、そのまま利用し続けることは得策ではない。たとえば、利用者数の増加や新たな攻撃に対処できないソフトウェアをそのまま利用し続けることは、システム障害や情報漏洩などの問題を引き起こす。このため、近年のソフトウェア開発では、その保守において継続的に変更を適用し、信頼性や安全性を維持・向上させることが必須である。

このような状況において、ソフトウェアが過去にどのように作成あるいは変更されたのかを知ることは、将来のソフトウェア保守において重要である。たとえば、ある要求に基づき過去に実施された変更を十分に把握せずに、別の要求に対する変更で上書きしてしまうと、新たな不具合（実行速度の低下やセキュリティホール）の混入につながる恐れがある。

このような背景から、プログラムの理解モデルやソフトウェア進化モデルの構築だけではなく、プログラムコードの変化（コードの追加、修正、改善など）を抽出、分析、活用する手法やツールの研究も活発になってきていた。また、オープンソースソフトウェアの台頭により、過去のソフトウェア開発に関する大量の実データが研究素材として容易に利用可能となっていた。これを受け、過去のプログラムコードの変化やそれに付随する情報をリポジトリに保管し、それを将来の開発・保守支援に積極的に活用する研究が盛んになっていた。

このように、プログラム理解やソフトウェア進化という観点から、プログラムコードの変化を取り扱う研究は活発になっていた。また、ソフトウェアの延命化による保守コストの増大から、過去に実施されたプログラム変更を把握する手法やツールの確立に対する社会的要請も大きかった。しかしながら、現実の開発・保守現場で利用可能なプログラム変更支援ツールはまだまだ貧弱であった。たとえば、多くの開発・保守現場では過去のソースコードの版管理ツールが導入されているが、このツールでは版間のソースコードの差分がプログラム変更情報として提示されるだけであった。また、ソースコードの修正を管理する目的のためにバグ追跡システムが存在するが、このシステム単体ではバグの修正、さらにはバグの混入を防御するような支援は行えていなかった。将来のソフトウェア保守に役立つプログラム変更情報を抽出するためには、単純にソースコードの差分を提示するだけでは不十分であり、プログラム変更に関わるさまざまな情報を記録、追跡、分析、検索できるツールが必須である。さらに、このようなツールを容易に作成可能な環境を提供することも重要である。

２．研究の目的

本研究では、開発者や保守者が過去に行ったプログラムのソースコードの編集操作（文字の挿入や削除のような小さなコード編集から、リファクタリングのようなメニュー選択による大きなコード変換まで）をすべて記録し、その編集操作履歴から将来のプログラム保守に役立つ情報を抽出する手法を確立する。

さらに、その情報に基づきプロアクティブに（先を見越して）プログラム変更をナビゲートするソフトウェア開発・保守環境の構築を目指す。ナビゲートとは、要求や環境の変化に応じたプログラムの改変候補の提示、品質向上のためのプログラム改善候補の提示、危険なプログラム変更に対する警告、プログラム理解支援のための視覚化など、開発者や保守者の作業に指示を与えることを指す。

本研究では、次に示す３つの研究項目の実現を目指す。

（１）過去のソフトウェア開発・保守におけるソースコードの編集操作を記録および追跡する仕組みの確立と、記録した編集操作履歴を効率的に扱うための内部表現の決定

（２）プログラム変更をナビゲートするツール群の構築と、従来のソフトウェア統合開発環境とのシームレスな連携の実現方法の確立

本研究では、ソフトウェア保守における変更を、詳細設計や実装レベルにおけるプログラム変更（ソースコード変更）に限定する。これは、実際のソフトウェア保守現場において、プログラムが唯一信用できる成果物である点、変更が要求仕様書や設計仕様書に反映されていないという状況が頻繁に発生している点を考慮した結果である。

３．研究の方法

従来の研究では主に、リポジトリに保管された２つの版のソースコードの差分からプログラム変更を抽出していた。これに対して、本研究では、ソースコードの編集操作を細粒度で記録し、記録した編集操作履歴からプログラム変更を追跡・分析することが大きな特徴である（図 1）。

編集操作を細粒度に記録・追跡・分析することで、開発者や保守者が実施した変更を正確に把握できる可能性が高まり、開発者や保守者の意図や状況に応じたプログラム変更支援が期待できる。また、追跡情報が付与された編集操作履歴を容易に活用可能なツールプラットフォームを提供することで、新た

なプログラム変更支援ツールの登場を促進することができる。

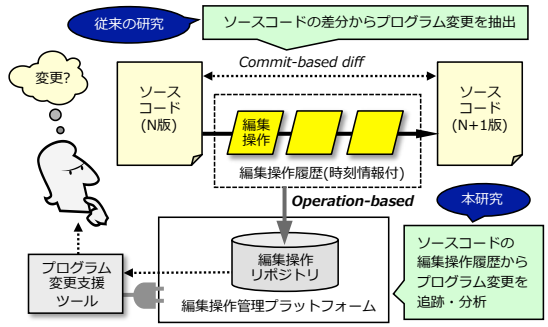

図 1: 本手法の概要と特徴

各研究項目に対する取り組みを次に示す。

（1）過去の編集を追跡する仕組みの確立と履歴の内部表現を決定する。このために、統合開発環境 Eclipse の Java エディタ上で実施されたすべての編集操作と、メニュー操作を記録するツール OperationRecorder を拡張する。このツールにおいて、編集操作履歴は、過去の編集操作を単純に時刻順に並べたものである。本研究では、編集操作履歴を効率的に活用するために、ツール間で共有可能な内部表現を定義する。

（2）将来のプログラム変更を支援するという観点から、編集操作履歴を用いることで、過去のプログラム変更の把握を支援するツールを作成する。

4．研究成果

主に 4 つの研究成果を達成した。

（1）従来研究において開発した、編集操作記録ツール OperationRecorder の記録機能を改良した。このツールは、統合開発環境 Eclipse の Java エディタ上でプログラマが行ったソースコードの編集操作を細粒度で記録できる。しかしながら、編集操作の記録は Java エディタ上のファイルに基づき管理されており、ファイル名の変更などを正確に追従できない。そこで、Java エディタ上で行われたものだけを記録するのでなく、リソースの変更操作（ファイルの削除、移動、ファイル名の変更）や取り消し（undo/redo）操作を記録できるようにツールを拡張した。リソースの変更や取り消し操作を記録することで、編集操作の再演時において編集操作が正確に追従できるようになったことを確認した。

（2）ソフトウェア開発時のプログラム変更情報を検出し保守者に提示することで、保守者によるプログラムの変更過程の理解を促進できる。このため、ソースコードの 2 つの版から行差分を抽出する手法が提案されている。しかしながら、単一の差分情報に目的の異なる複数の変更が混ざり合っている場合、これらの手法では変更を適切に分離できない。このため、保守者は混ざり合った変更を人手で分離し、個々の変更内容を推測しなくてはならず、その負担は大きい。

そこで、OperationRecorder により記録される編集操作履歴を用いることで、過去におけるソースコードの任意の時点でのスナップショットを機械的に復元し、連続する 2 つのスナップショット間に実施されたプログラム変更を自動検出する手法を確立した。さらに、検出された変更を開発者や保守者の方針（時間的距離と空間的距離に関する基準）に応じて集約する仕組みを確立した。プログラム変更の検出と集約手法を実装したツール（図 2）を用いた評価実験により、時間的あるいは空間的に集約されたプログラム変更が保守者のプログラム理解活動を支援できることを示した。

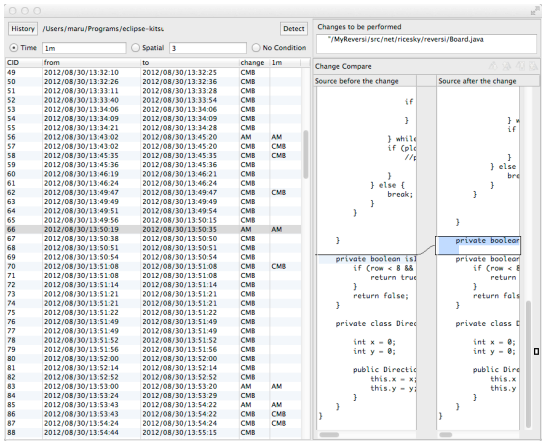

図 2: プログラム変更検出・集約ツール

（3）編集操作記録ツールにより記録した編集操作履歴において、連続する 2 つの編集操作間に存在する依存関係を、それぞれの編集位置（ソースファイルにおける編集箇所のオフセット値の計算）に基づき抽出する手法を確立し、そのような関係を形式的に表現する編集操作グラフを構築した。このグラフ表現では、それぞれの編集操作の適用前後のソースコードのスナップショットに対して、クラスメンバを節点として、それらの節点間の編集に関する依存関係を表す 4 種類の辺が定義されている。また、グラフ表現の節点をクラスメンバから編集操作に変更することで、開発者の知りたい情報や開発状況に対して、より柔軟な過去の編集操作の追跡性が実現できた。

編集操作グラフを活用することで、特定のクラスメンバの作成に関係する編集操作だけを抽出する編集操作スライシング手法を確立し、編集操作の再演ツールに組み込んだ（図 3 と図 4）。編集操作を記録した 3 つのアプリケーション開発（3 名の被験者による）を対象とした評価実験を行うことで、プログラム理解コストが削減できることを確認した。

（4）統合開発環境において記録した細粒度な編集操作履歴の新たな活用方法を探し出

すために、編集履歴を収集する手法および編集履歴を応用する手法の調査を行った。

調査を通して、編集操作履歴を積極的に活用することで、従来の改版履歴に基づく手法において発生する多くの問題点が解決可能であることが分かった。同時に、現時点で提案されている手法の利用環境は限定的であること、実際の開発現場や保守現場における適用事例が少ないことが判明した。

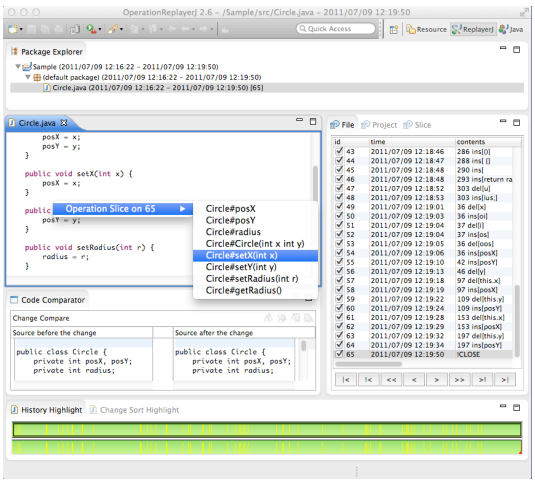

図 3: 編集操作スライスの抽出

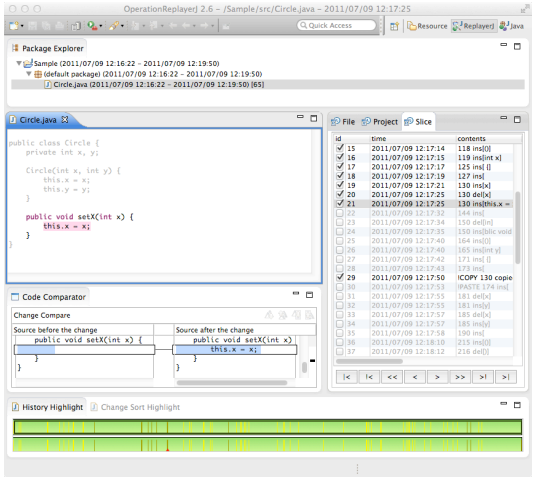

図 4: 編集操作スライスの再演

5．主な発表論文等

〔雑誌論文〕（計 9 件）

① Katsuhisa Maruyama、Takayuki Omori、Shinpei Hayashi、Slicing Fine-Grained Code Change History、IEICE Transactions on Information and Systems、査読有、Vol.E99-D、No.3、pp.671-687、2016
DOI: 10.1587/transinf.2015EDP7282

② Rully Agus Hendrawan、Katsuhisa Maruyama、Visualizing Time-based Weighted Coupling Using Particle Swarm Optimization to Aid Program Comprehension、Procedia Computer Science、査読有、Vol.72、pp.597-604、2015
DOI: 10.1016/j.procs.2015.12.168

③ 木津栄二郎、大森隆行、丸山勝久、コードの編集履歴を用いたプログラム変更の検出、情報処理学会論文誌、査読有、Vol.56、No.2、pp.611-626、2015
http://id.nii.ac.jp/1001/00113142/

④ 大森隆行、林晋平、丸山勝久、統合開発環境における細粒度な操作履歴の収集および応用に関する調査、コンピュータソフトウェア、査読有、Vol.32、No.1、pp.60-80、2015
DOI: 10.11309/jssst.32.1_60

⑤ Takayuki Omori、Hiroaki Kuwabara、Katsuhisa Maruyama、Improving Code Completion based on Repetitive Code Completion Operations、コンピュータソフトウェア、査読有、Vol.32、No.1、pp.120-135、2015
DOI: 10.11309/jssst.32.1_120

⑥ Rizky Januar Akbar、Takayuki Omori、Katsuhisa Maruyama、Mining API Usage Patterns by Applying Method Categorization to Improve Code Completion、IEICE Transactions on Information and Systems、査読有、Vol.E97-D、No.5、pp.1069-1083、2014
DOI: 10.1587/transinf.E97.D.1069

⑦ 丸山勝久、酒屋問題再考 -新たな共通問題の作成を目指して-、情報処理学会情報処理、査読無、Vol.54、No.9、pp.886-889、2013
https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_uri&item_id=94769&file_id=1&file_no=1

⑧ 木津栄二郎、大森隆行、丸山勝久、ソースコード編集履歴を用いたプログラム変更の検出、コンピュータソフトウェア、査読有、Vol.29、No.2、pp.168-173、2012
DOI: 10.11309/jssst.29.2_168

⑨ 大森隆行、丸山勝久、林晋平、沢田篤史、ソフトウェア進化研究の分類と動向、コンピュータソフトウェア、査読有、Vol.29、No.3、pp.3-28、2012
DOI: 10.11309/jssst.29.3_3

〔学会発表〕（計 11 件）

① Shinpei Hayashi、Daiki Hoshino、Jumpei Matsuda、Motoshi Saeki、Takayuki Omori、Katsuhisa Maruyama、Historef: A Tool for Edit History Refactoring、22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)、pp.469-473、2015 年 3 月 3 日、Montreal (Canada)

② 田島香織、丸山勝久、ネスト化によるリファクタリングの連続的適用、日本ソフトウェア科学会ソフトウェア工学の基礎ワークショップ (FOSE2014)、pp.225-230、

2014年12月13日、霧島国際ホテル（鹿児島霧島市）

③ Katsuhisa Maruyama、Takayuki Omori、Shinpei Hayashi、A Visualization Tool Recording Historical Data of Program Comprehension Tasks、22nd International Conference on Program Comprehension (ICPC 2014)、pp.207-211、2014年6月3日、Hyderabad (India)

④ 田島香織、大森隆行、丸山勝久、一時変数除去の自動化によるメソッドの抽出リファクタリング支援、電子情報通信学会ソフトウェアサイエンス研究会、2013年7月26日、北海道立道民活動センター（北海道札幌市）

⑤ Eijirou Kitsu、Takayuki Omori、Katsuhisa Maruyama、Detecting Program Changes from Edit History of Source Code、20th Asia-Pacific Software Engineering Conference (APSEC'13)、2013年12月4日、pp.299-306、Bangkok (Thailand)

⑥ 宮本崇史、丸山勝久、GUIプログラムにおけるイベント処理の可聴化によるデバッグ支援、ソフトウェア工学の基礎ワークショップ(FOSE2014)、pp.239-244、2013年11月30日、ゆのくに天祥（石川県加賀市）

⑦ 丸山勝久、ソフトウェア進化との付き合い方 -変化を受け入れるソフトウェア開発・保守技術-、JEITA ソフトウェアエンジニアリング技術ワークショップ2012、2012年12月14日、ＴＫＰ大手町カンファレンスセンター(東京都千代田区)

⑧ 丸山勝久、ソフトウェア進化の研究、ソフトウエア・メインテナンス・シンポジウム2012、2012年10月15日、全国情報サービス産業厚生年金基金会館（東京都中央区）

⑨ Shinpei Hayashi、Takayuki Omori、Teruyoshi Zenmyo、Katsuhisa Maruyama、Motoshi Saeki、Refactoring Edit History of Source Code、28th IEEE International Conference on Software Maintenance (ICSM 2012)、pp.617-620、2012年9月27日、Trento (Italy)

⑩ Takayuki Omori、Hiroaki Kuwabara、Katsuhisa Maruyama、A Study on Repetitiveness of Code Completion Operations、28th IEEE International Conference on Software Maintenance (ICSM 2012)、pp.584-587、2012年9月25日、Trento (Italy)

⑪ Katsuhisa Maruyama、Eijiro Kitsu、Takayuki Omori、Shinpei Hayash、Slicing and Replaying Code Change History、27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)、pp.246-249、2012年9月5日、Essen (Germany)

６．研究組織

(1)研究代表者

丸山 勝久 (MARUYAMA KATSUHISA)
立命館大学・情報理工学部・教授
研究者番号： 30330012

(2)研究分担者

大森 隆行 (OMORI TAKAYUKI)
立命館大学・情報理工学部・任期制講師
研究者番号： 90532903

(3)連携研究者

林 晋平 (HAYASHI SHINPEI)
東京工業大学・情報理工学研究科・助教
研究者番号： 40541975