様式 C-19

科学研究費補助金研究成果報告書

平成 23 年 6 月 1 日現在

機関番号：12608
研究種目：研究活動スタート支援
研究期間：2009～2010
課題番号：21800018
研究課題名（和文）　ソースコード変更に基づくソフトウェア分析環境の構築

研究課題名（英文）
Study on Software Analytic Environment Based on Source Code Changes

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

研究成果の概要（和文）：
本研究では，ソフトウェアの保守や改良などの際におけるソースコードの理解と修正を円滑とするために，該当ソフトウェア開発プロジェクトで過去に行われたソースコード変更の理解及び分析を容易とするための技術を開発し，その作業を支援する自動化ツールを実現した．

研究成果の概要（英文）：
In this research project, we have developed some techniques for understanding and analyzing source code changes in order to reduce the costs for understanding and modifying source code in a software maintenance process. Furthermore, we have implemented computerized tools for supporting developers' activities based on the techniques above.

交付決定額

（金額単位：円）

	直接経費	間接経費	合　計
2009 年度	960,000	288,000	1,248,000
2010 年度	750,000	225,000	975,000
年度			
年度			
年度			
総　計	1,710,000	513,000	2,223,000

研究分野：工学
科研費の分科・細目：ソフトウエア
キーワード：ソフトウェア変更，ソフトウェア構成管理，ソフトウェア理解，リファクタリング

１．研究開始当初の背景

ソフトウェア変更の理解や分析は重要である．ソフトウェアは一度構築された後も，要求変化への対応，環境変化への適応，バグフィクス，将来のリリースや引き継ぎのための可読性や拡張性の向上など，様々な変更が行われる．こういった変更の際には，対象ソフトウェアの振る舞いや開発経緯を良く理解することが求められる．しかし，この理解は簡単ではない．特にオープンソースソフトウェア（OSS）など，多くの開発者が関わり，変更が多発するソフトウェア開発プロジェクトではこれが顕著である．頻繁に変更されるソフトウェアでは，開発者が理解していた振る舞いが後の変更により変わってしまい，新たなる理解が求められる．変更により生まれた理解のギャップを埋めるためには，変更そのものを低コストで理解することが有効である．

２．研究の目的

本研究では，１．で挙げた背景に基づき，変更の理解や分析を促進すべく以下の課題に取り組んだ．

(1) 抽象度の高い変更の分離：ソースコード変更はさまざまな抽象度で表現することができ，またそれが理解の容易さに強く影響する．特に，リファクタリングをはじめとした，抽象度の高い変更の表現とその名前がイディオムとしてよく知られており，ソースコードに行われた変更の大部分を，こういった抽象度の高い変更として表現し，その情報を利用することができれば，変更の理解が容易となる．

(2) 変更の追跡：ある変更が過去のどの時点で起こったかを特定したり（origin analysis），過去から現在に至るまでの一連の変更を分析したりするためには，旧版と新版のソースコードの各要素における対応関係を特定する必要がある．また，特定のソフトウェア機能に対応するコードの変更を追跡する場合は，機能とコードとの対応付けも必要となる．

これらが実現されることで，日々変更されるソースコードに開発者が理解を追従させることを支援できる．

３．研究の方法

本研究では，２．(1) を実現するため，抽象度の高い変更としてリファクタリング操作に注目し，その分離を試みた．リファクタリングとそれ以外の変更を分離するため，すでにそれらが混在した変更による差分からリファクタリング操作を探索的に抽出し，分離を行う後ろ向きのアプローチと，複数の意図の変更を行う際に，現在行っている意図を表明しながらソースコード変更を行うことにより，混在のない形で変更を記録可能な開発環境を利用する前向きのアプローチの双方を行った．

また，２．(2) の実現のために，時系列上でのコード片の対応付け及びコード片と機能との対応付け手法を開発した．まず，旧版と新版のソースコードの要素を細粒度で対応付け，ソースコード片の各要素における対応関係を特定する手法を検討した．さらに，特定のソフトウェア機能に対応するコードの変更を追跡する場合は，機能とコードとの対応付けも必要となるため，ソフトウェア機能とソースコードの対応付けとして，機能を表す自然言語記述からコード片を自動抽出する Feature Location 技術を開発した．いずれの手法も，開発者や外部リソースが持つ知識や試行錯誤の経験を反映できるよう，対話的な環境として実現した．

いずれのアプローチにおいても，広く用いられている Java 言語を利用して構築されたオブジェクト指向プログラムを対象とした支援ツールを開発した．

４．研究成果

(1) リファクタリングの抽出・適用による差分からの分離：リファクタリングと通常の変更が混在したソースコードの差分からリファクタリングの影響を分離し，振る舞いの理解に適した差分を得る手法を開発した．手法の概要を図１に示す．この手法では，まず与えられた旧版のソースコード old 及び新版のソースコード new の比較により得られた差分からリファクタリング操作を抽出する．次にそれらを一方の版のソースコードに実際に適用して新たなソースコード modified を得，これを比較に用いることにより，リファクタリングの影響を除外した差分Δを得る．提案手法では，リファクタリングの抽出や適用の順序・粒度の規則を定め，パッケージレベルからクラスレベル，メソッドレベルへと，段階的にリファクタリングを抽出可能とした．また，Rename Method をはじめとする 10 種類のリファクタリングの適用操作を，Eclipse のリファクタリングブラウザを用いて自動化し，制御可能とした．

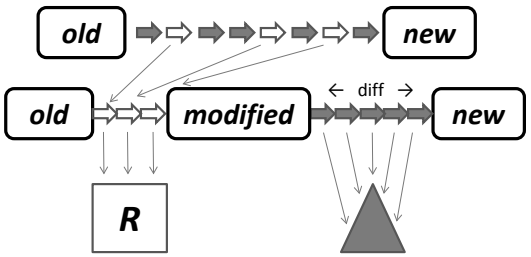

図１：リファクタリング分離手法の概要

理解に適した差分を得る支援ツールを開発した．オープンソースソフトウェア OW2 Carol の 38 リリース，Apache Tomcat の 24 リリースを対象とし，各リリース間に提案手法を適用したところ，平均で全体行数のうち 21.4% の差分，またリファクタリング操作１つにつき平均 31 行の差分が削減されることがわかった．

(2) 編集操作の分類によるソースコード差分の再構成：ソースコードの編集操作履歴に対して注釈付けを行うことにより，複数の意図に基づいた変更が混在した編集操作を，理解に適するよう意図のまとまりごとの形に再構成して版管理リポジトリにコミットする手法及びその支援ツールを開発した．提案手法の概要を図２に示す．この手法では，開発者は変更の意図を注釈付けながら編集が行える特別なエディタを用いてソースコードを変更する．一連の編集作業が終わった後，行われた編集内容を変更意図ごとに並べ替え，個別に版管理リポジトリにコミットすることができる．編集操作の並べ替え規則及び並べ替えが行えるかの判定をリアルタイムに行えるアルゴリズムを定義し，手法の自動化を可能とした．また，編集操作記録ツール

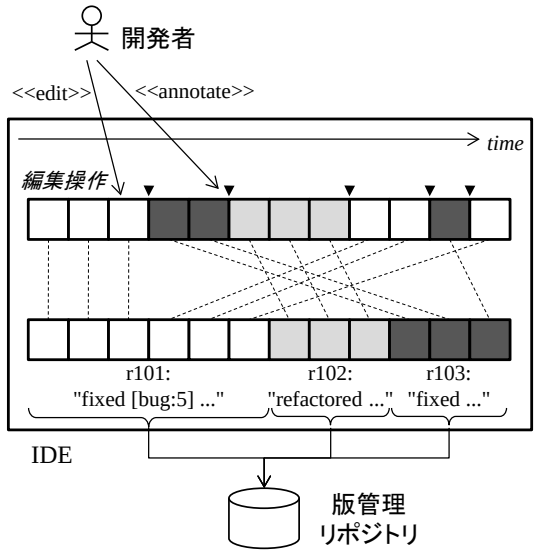

図 2：編集操作分類手法の概要

OperationRecorder を利用し，Eclipse 上に手法を実現するツールを開発した．このツールにより，リファクタリングやバグ修正など複数の意図に基づく変更を再構成し，個別にコミットできることを確認しており，既存の版管理システムと親和性の高い形で，理解に適したソースコード変更を低コストで記録することが可能となった．

（3）コード片の対応関係の特定：版管理履歴における特定の版のコード片に対応する過去の版のコード片を細粒度で特定する対話的な手法を検討した．この手法では，コード片をコード行の集まりとしてとらえ，対応する一連の過去の版におけるコード行の集まりを特定する．低コストで正しい対応付けを得るため，機械的に得られた特定結果の不対応や誤対応の誤りを指摘することより結果を対話的に修正できる．この手法により，少なくとも Apache Hadoop リポジトリから得たコード片例に限っては，誤検出の対応関係の指摘により正しい対応関係を得られたことが確認できた．

（4）機能の自然言語記述に対応するソースコードの特定：ソースコード変更の理解の際に，変更箇所に関連する仕様書上の記述を特定するため Feature Location（FL）手法を 2 つ開発し，それぞれ支援ツールを構築した．

まず，ドメインオントロジを用いて対応関係を構成する FL 手法を開発した．この手法では，自然言語記述とソースコード上の識別子との類似性に基づく関係と，コード上のメソッド呼び出し関係の評価にドメインオントロジによる意味的関係を考慮することで，詳細ではない記述に対しても高精度の対応付けを行う．まず，ソースコードの静的解析により，メソッドを節とするコールグラフを

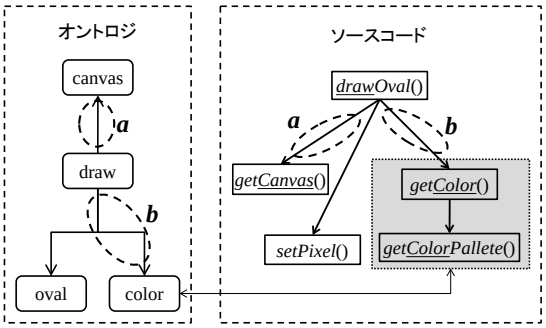

図 3：オントロジの利用例

構築する．入力する自然言語とソースコード上の識別子のマッチングにより機能を代表するメソッドの候補を特定し，この候補からメソッド呼び出しを辿ることによりコールグラフの部分グラフを FL 結果の候補として得る．どのメソッド呼び出しを辿るかに，あらかじめ用意したドメイン知識を蓄えたドメインオントロジを用いることにより，精度良い結果を得る．図 3 にオントロジの利用例を示す．この例では，図形描画ソフトウェアのオントロジを用いており，これには「キャンバス（canvas）が描画（draw）対象となる」等の知識が関係として表現されている．ソースコード中のメソッド呼び出し関係とオントロジの関係が共起した場合，該当のメソッド呼び出しを重要なものとして FL 結果に含めている．オープンソースソフトウェア JDraw の提供するマニュアルに記載された 7 機能を対象とし，提案手法を適用したところ，うち 5 機能で検出精度がオントロジを利用しなかった場合に比べ向上したことがわかった．

さらに，利用者からのフィードバックをもとに特定結果を改善し，対応するコード片を効率的に発見する FL 手法も開発し，その支援ツール iFL を構築した．この方式では，入力した機能記述をクエリとし，FL 結果としてメソッド呼び出しイベントを優先順位付きで利用者に提示する．図 4 に支援ツールの画面例を示す．この例では，利用者がクエリ「schedule」を用いて FL を行っており，結果の最上位にメソッド addSchedule の呼び出しイベントがある．提案手法では，利用者は高位にランクされたイベントに対応するメソッドを読み，その結果をシステムにフィードバックする．すなわち，該当イベントが想定した機能と関係があれば要，関係がなければ不要のヒントをシステムに与える適合フィードバックを行う．システムは，これらの情報を優先順位計算に反映させる．提案手法では，解析対象のプログラムの静的・動的解析結果をもとに優先順位を決定している．基本的には，解析により得た動的依存グラフのうち，要のイベントの近辺にあるイベントの順位が上がり，不要のイベントの近辺の順位

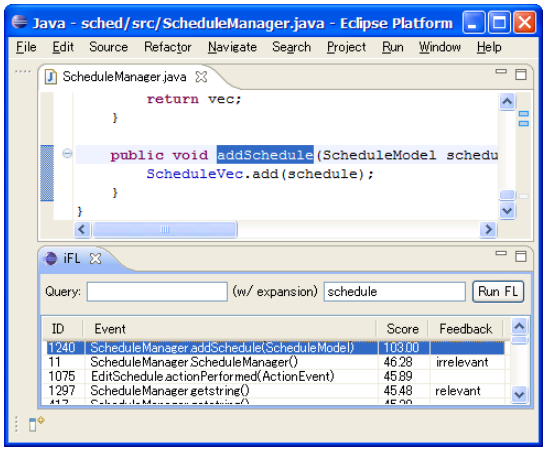

図 4：支援ツール iFL の動作例

が下がるようになっている．スケジュール管理ソフトウェア及び JDraw の計 7 機能に対して提案手法を適用したところ，うち 6 機能で提案手法を利用しなかった場合と比べ低コストで理解プロセスを終了できることがわかった．

５．主な発表論文等
（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計 7 件）

① Shinpei Hayashi, Takashi Yoshikawa, Motoshi Saeki, Sentence-to-Code Traceability Recovery with Domain Ontologies, Proceedings of the 17th Asia Pacific Software Engineering Conference (APSEC 2010), 385-394, 2010, 査読有

② Shinpei Hayashi, Motoshi Saeki, Recording Finer-Grained Software Evolution with IDE: An Annotation-Based Approach, Proceedings of the 4th International Joint ERCIM/IWPSE Symposium on Software Evolution (IWPSE-EVOL 2010), 8-12, 2010, 査読有

③ Shinpei Hayashi, Katsuyuki Sekine, Motoshi Saeki, iFL: An Interactive Environment for Understanding Feature Implementations, Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM 2010), 1-5, 2010, 査読有

④ 小林隆志, 林晋平, データマイニング技術を応用したソフトウェア構築・保守支援の研究動向, コンピュータソフトウェア, vol. 27, no. 3, 13-23, 2010, 査読有

⑤ 林晋平, 佐々木祐輔, 佐伯元司, 階層分析法を応用したソースコード変更案の評価, コンピュータソフトウェア, vol. 27, no. 2, 118-123, 2010, 査読有

⑥ Shinpei Hayashi, Yasuyuki Tsuda, Motoshi Saeki, Search-Based Refactoring Detection from Source Code Revisions, IEICE Transactions on Information and Systems, vol. E93-D, no. 4, 754-762, 2010, 査読有

⑦ Takashi Yoshikawa, Shinpei Hayashi, Motoshi Saeki, Recovering Traceability Links between a Simple Natural Language Sentence and Source Code Using Domain Ontologies, Proceedings of the 25th International Conference on Software Maintenance (ICSM 2009), 551-554, 2009, 査読有

〔学会発表〕（計 5 件）

① 林晋平, 変更履歴のリファクタリングに向けて, 情報処理学会ソフトウェア工学研究会ウィンターワークショップ 2011・イン・修善寺, 2011 年 1 月 21 日, ラフォーレ修善寺

② 関根克幸, 林晋平, 佐伯元司, Feature Location を用いたソースコード理解の対話的支援, ソフトウェアエンジニアリングシンポジウム 2010, 2010 年 9 月 1 日, 東洋大学白山キャンパス

③ タンタムマチットシリナット, 林晋平, 佐伯元司, リファクタリングの抽出・適用によるソースコード差分の理解支援, ソフトウェアエンジニアリングシンポジウム 2010, 2010 年 9 月 1 日, 東洋大学白山キャンパス

④ 林晋平, 佐伯元司, 編集操作の分類に基づくソースコード差分の構造化, 電子情報通信学会ソフトウェアサイエンス研究会, 2010 年 3 月 8 日, 鹿児島大学郡元キャンパス

⑤ 林晋平, 時系列への注釈に基づくソースコード変更の再構成, 情報処理学会ソフトウェア工学研究会ウィンターワークショップ 2010・イン・倉敷, 2010 年 1 月 21 日, 倉敷市芸文館

６．研究組織
(1)研究代表者

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