修士学位論文

題目

コピーアンドペーストを考慮した Edit Script の生成

指導教員
楠本 真二 教授

報告者
大谷　明央

平成 28 年 2 月 9 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻

平成 27 年度 修士学位論文

コピーアンドペーストを考慮した Edit Script の生成

大谷　明央

内容梗概

ソフトウェア進化の分野において，ソフトウェアがどのように変更されるかをよりよく理解するための研究が広く行われている．ソフトウェアの変更内容を理解することは，近年広く用いられているバージョン管理システムにおける，システムの管理者や，コードレビューを行う開発者達にとって重要である．そのため，ソフトウェアの変更内容を開発者にとって理解しやすく表現することはソフトウェア開発において重要な課題であり，これまでに多くの手法が提案されてきた．その中でも，抽象構文木に基づく手法は，プログラムの構造を考慮するため，開発者にとって理解しやすく変更内容を表現できる．抽象構文木に基づく手法では，Edit Script を生成することにより変更内容を表現する．既存の抽象構文木に基づく手法では挿入，削除，更新，移動といった 4 つの操作を考慮して Edit Script を生成する．

本研究では，既存の抽象構文木に基づく手法を改良することにより，開発者にとってより理解しやすく変更内容を表現する手法を提案する．既存手法で考慮されている操作の他にも，ソフトウェア開発において頻繁に行われる操作として，コピーアンドペーストが挙げられる．そのため，コピーアンドペーストを考慮して Edit Script を生成することにより，開発者にとってより理解しやすく変更内容を表現できる．提案手法を評価するために，複数のオープンソースソフトウェアに対して調査を行った．その結果，提案手法が既存手法と比較して開発者にとってより理解しやすい Edit Script を生成することを確認した．

主な用語

ソフトウェア進化
プログラム理解
Edit Script
抽象構文木
コードクローン

目 次

図 目 次

表 目 次

1　はじめに

ソフトウェア進化の分野において，ソフトウェアがどのように進化するかをよりよく理解するための研究が広く行われている [1]．ソフトウェア進化には，要件や実行環境の変更といったグローバルな進化と，ソースファイルやドキュメント等の変更といったローカルな進化がある．本論文ではローカルな進化に焦点を当て，ソースファイルの変更内容を表現する手法を研究する．

近年，ソフトウェア開発においてバージョン管理システムが広く用いられている [2][3][4]．バージョン管理システムにおいて，システムの管理者や，コードレビューを行う開発者達は，ソースファイルに対して行われた変更の内容を理解する必要がある．そのため，ソースファイルの変更内容を開発者にとって理解しやすく表現することはソフトウェア開発において重要であり，これまでに多くの研究が行われている．

ソースファイルの変更内容を表現する手法のうち広く使われるものに，テキストに基づいて変更内容を表現する手法がある [5][6][7]．例えば，diff では，入力としてソースファイルを 2 つ与えると，Myers のアルゴリズム [5] を適用し，行の追加や削除を示して変更内容を表現する．しかし，テキストに基づく手法には，プログラムの構造を考慮することができないという弱点がある．プログラムの構造を考慮しないため，表現された変更内容が開発者にとって理解しやすいものになるとは限らない．

テキストに基づく手法の弱点を克服するために，抽象構文木に基いて変更内容を表現する手法がある [8][9][10][11]．抽象構文木に基づく手法は，プログラムの構造を考慮して変更内容を表現することができるため，開発者にとってより理解しやすく変更内容を表現することができる．抽象構文木に基づく手法では，Edit Script を生成することにより，プログラムの構造を考慮して変更内容を表現する．Edit Script とは，2 つの抽象構文木が与えられたとき，一方の抽象構文木をもう一方の抽象構文木に変更するための操作の手順を表したものである [11]．Edit Script において，変更内容は，抽象構文木におけるノードへの操作の連なりとして表現される．変更において行われる操作が多いほど，開発者が変更内容を理解する負担が大きくなるため，Edit Script の長さによって開発者が変更内容を理解する際の労力を表すことができる [12]．そのため，生成される Edit Script が短いほど開発者にとって変更内容を理解しやすいといえる．

抽象構文木に基づく手法には，非常に多くのノードを含む抽象構文木を対象にした場合や，ノードの移動を考慮した場合において計算に長い時間を要するという問題点がある．こうした問題を解決するために，Falleri らの手法 [12] では，Edit Script の生成の過程で経験則を用いることで，大きな抽象構文木に対しても効率的に移動を考慮した Edit Script を生成することができる．また，Falleri らは実際のソフトウェア開発に用いられたリポジトリを

対象に実験を行い，Falleri らの手法が実用的な速度で動作することを示し，また他の手法と比較することで，生成する Edit Script が開発者にとってより理解しやすいといえることを示した．

本研究では，既存の手法よりもさらに開発者にとって理解しやすく変更内容を表現することを目的として，Falleri らの手法を改良した．Falleri らの手法では挿入，削除，移動，更新を組み合わせて変更内容を表現する．しかし，実際のソフトウェア開発において，この 4 つの操作の組み合わせでは変更内容をうまく表現できない場合がある．Falleri らの手法を用いてコピーアンドペーストを Edit Script で表すと，新しいノードが多数挿入されたと表現される．しかし，コピーアンドペーストによって作成されたコードは実際には全く新しいものではないため，既存のコードを用いて表現できる．ソフトウェア開発において，コピーアンドペーストが頻繁に行われることがわかっている [13][14]．そこで本研究では，コピーアンドペーストを考慮することにより，開発者にとってより理解しやすく変更内容を表現する手法を提案する．

また，本研究では，提案手法を実装し，手法の有効性を評価するための実験を行った．実験では，14 個のオープンソースソフトウェアのリポジトリを含む CVS-Vintage dataset を対象とし，13,699 個の変更を対象に提案手法と Falleri らの手法を比較した．調査の結果，提案手法を用いることにより開発者にとって理解しやすい Edit Script を生成できることを確認した．以下に，実験によって得られた結果をまとめる．

- 18%の変更に対して提案手法が生成する Edit Script は Falleri らの手法が生成する Edit Script より短い．
- Falleri らの手法が生成する Edit Script が提案手法が生成する Edit Script より短くなる変更は存在しない．
- 提案手法の実行時間は Falleri らの手法と比較して長くなるが，96%の変更において Falleri らの手法の 1.5 倍以内の時間で実行することができる．
- 96%の変更において提案手法は 2 秒以内に Edit Script を生成することができる．

以降，第 2 章では準備について述べる．第 3 章では研究の背景や既存研究について述べ，研究動機について説明する．第 4 章では提案手法の詳細について述べる．第 5 章では評価実験について，実験方法や結果について説明し，実験結果からの考察や実験の妥当性について述べる．最後に第 6 章では本研究のまとめについて述べる．

2　準備

2.1　ソフトウェア進化

ソフトウェア進化の法則で述べられるように，ソフトウェアは継続的に進化しなければならない [15]．そのため，ソフトウェアがどのように進化するかをよりよく理解するために，多くの研究が行われている [1]．ソフトウェア進化は，グローバルなソフトウェア進化とローカルなソフトウェア進化の 2 つに分類できる．

グローバルなソフトウェア進化:　要件や実行環境の変更など

ローカルなソフトウェア進化:　ソースファイルやドキュメントファイルの変更など

本研究ではローカルなソフトウェア進化，特にソースファイルへ行われる変更に焦点を当てる．

2.2　バージョン管理システム

ソフトウェア開発において，ソースファイルやドキュメント等のファイルへ行われる変更は，バージョン管理システムによって管理される．バージョン管理システムの例として，CVS[2]，Subversion[3]，Git[4] などが挙げられる．バージョン管理システムの概要を図 1 に示す．バージョン管理システムの主な機能として以下の 2 つが挙げられる．

- 開発情報の共有
  バージョン管理システムにおいて，ソースファイル等，ソフトウェアの開発情報はリポジトリと呼ばれるデータベースに保存される．開発者はソフトウェアを開発する際に，リポジトリから開発情報をコピーし，変更を加えた後，リポジトリに反映させる．複数の開発者が変更を行った場合にも，それらの開発情報は全て同一のリポジトリに保存される．このようにして，1 つのプロジェクトを複数の開発者で共有できる．

- 開発履歴の保持
  バージョン管理システムでは，開発履歴情報をリビジョンと呼ばれる単位で保存している．リビジョンとは開発の状態を示す単位であり，開発者が変更をリポジトリに反映させるたびに生成される．生成されるリビジョンにはリビジョン番号が割り当てられる．開発履歴情報にはソースファイルの変更といった開発情報に加えて，変更を行った開発者名やその日時，変更されたファイル名，変更を行った開発者による変更内容についてのメッセージなども含まれる．開発履歴情報を用いることで，ソースファイルを誤って変更してしまった際などに過去の状態に復元できる．

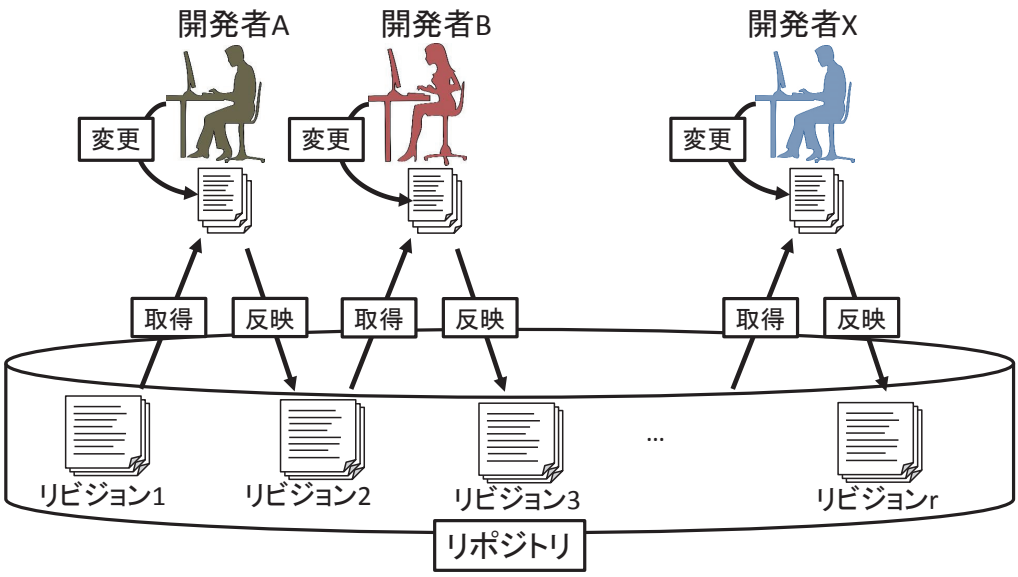


図 1: バージョン管理システム

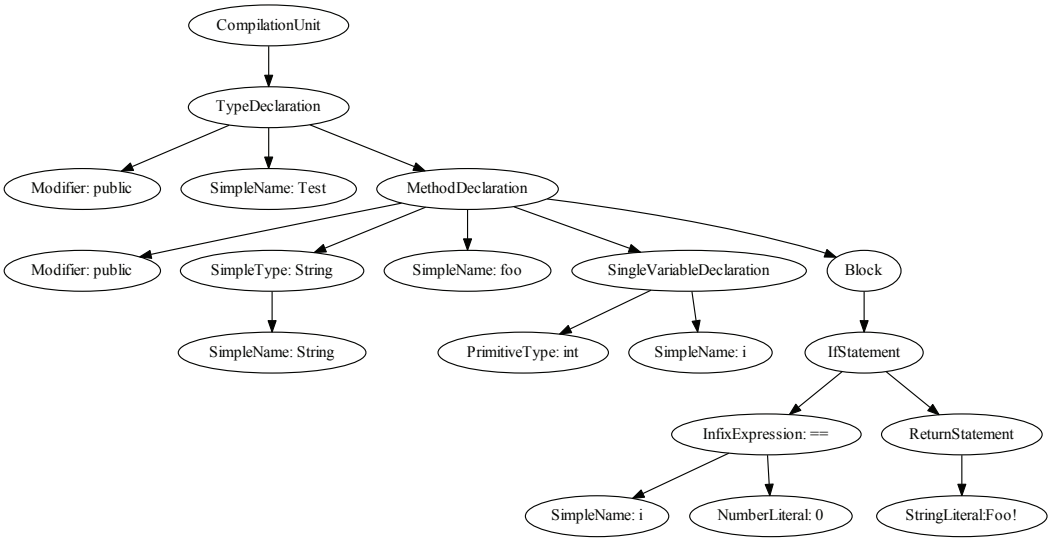
バージョン管理システムを用いることにより，離れた場所で開発を行う開発者の間で開発情報を容易に共有できるようになる．そのため，バージョン管理システムは複数の開発者が関わるソフトウェア開発において広く利用されている．

2.3 抽象構文木

抽象構文木（Abstract Syntax Tree，以下 **AST**）とは，木構造を用いてソースコードの構文情報を表現したものである．AST は順序木であり，子ノードの数に制限はない．例として，簡単な Java ソースコードとそれに対応する AST を図 2 に示す．この AST はプログラムの構造に対応する 19 のノードを持つ．AST のノードは，ID とソースコードの構造に対応するラベルとソースコード内の実際のトークンに対応する値を持つ．AST は構文解析木に対応した形をとる．AST のデータ構造はプログラミング言語に依存しないため，ソースコードの分析に適している．例えば，図 2b において「NumberLiteral:0」は NumberLiteral がラベル，0 が値を表す。AST 上のノードは構文上の 1 つの要素を表し，枝で直接結ばれた

```
public class Test{
 public String foo(int i){
   if(i == 0) return "Foo!";
 }
}
```

(a) ソースコード



(b) AST

図 2: AST の例

子ノードはその詳細情報を表す．例えば，図 2b において，IfStatement の構文情報は，その子ノードにより表され，その内容は If(InfixExpression) ReturnStatement であることが分かる．

2.4 コピーアンドペースト

ソフトウェア開発において，開発者はドキュメントや他人が書いたソースコードまたは自身が書いたソースコードといった様々な場所からソースコードの一部をコピーアンドペーストすることがある．ソフトウェア開発において，開発者がソースコードを頻繁にコピーアンドペーストすることが分かっている [13]．また，Ahmed らの研究により，開発者が行うコピーアンドペーストのうち，63.52%においてコピー元のファイルとコピー先のファイルが同一であることがわかっている [14]．コピーアンドペーストによって生成されたソースコードは，コピー元のソースコードとコードクローン関係になる．

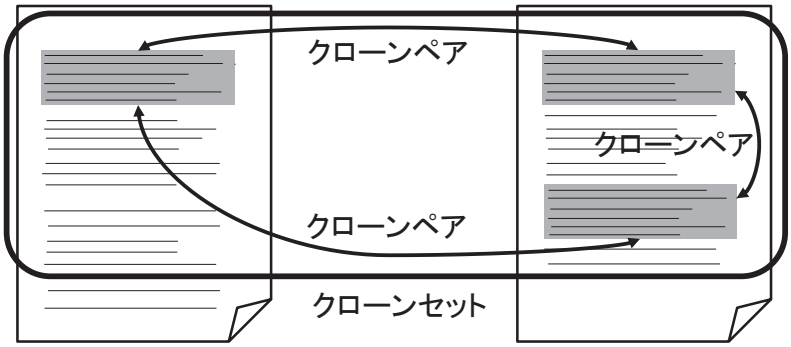


図 3: クローンペアとクローンセット

2.5 コードクローン

コードクローンとはソースコード中に存在する同一，あるいは類似するコード片のことである．コードクローンは既存ソースコードのコピーアンドペースによる再利用など，様々な理由で発生する [16][17][18]．図 3 に示すように，ソースコード中に存在する 2 つのコード片 α，β が類似しているとき，α と β は互いにクローンであるという．またペア（α，β）をクローンペアと呼ぶ．α，β それぞれを真に包含する如何なるコード片も類似していないとき，α，β を極大クローンと呼ぶ．また，互いにクローンであるコード片を同値としたときの同値類をクローンセットと呼ぶ [19]．ただし，どのような基準で類似していると判断するかは検出手法や検出ツールによって異なる [20][21]．

また，コードクローン間の類似の度合に基づきコードクローンを次の 3 つのタイプに分類することができる [22][23]．

Type-1 空白やタブの有無，括弧の位置などのコーディングスタイルに依存する箇所を除いて，完全に一致するコードクローン．

Type-2 変数名や関数名などのユーザ定義名，また変数の型などの一部の予約語のみが異なるコードクローン．

Type-3 Type-2 における変更に加えて，文の挿入や削除，変更が行われたコードクローン．

コードクローンを検出する手法はこれまでに多数提案されている．またそれらを実装した，コードクローンを自動的に検出するツールも多数開発されている [20][21].

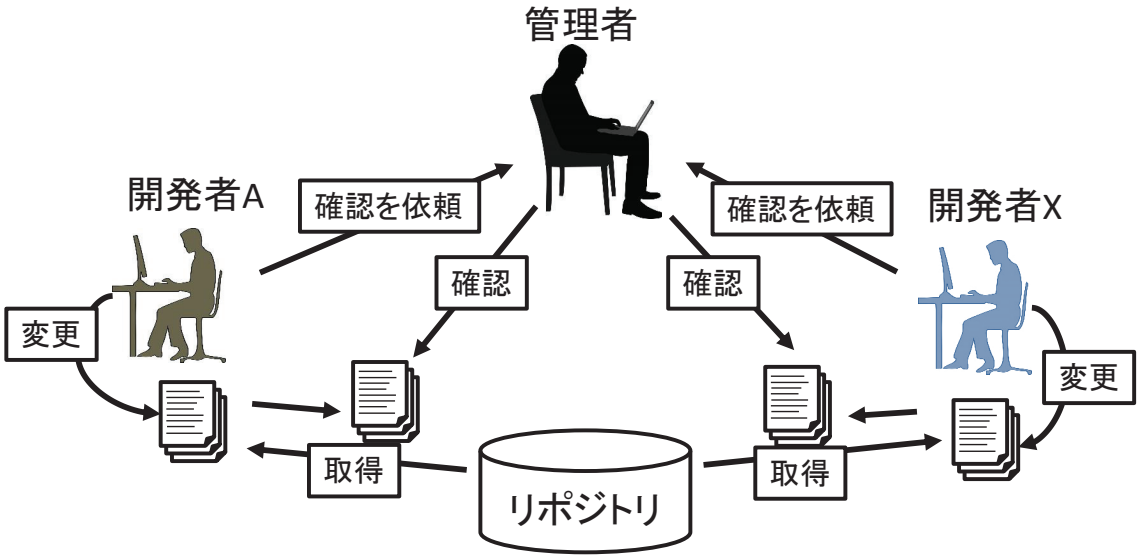


図 4: 管理者を置いたバージョン管理システム

3 背景

3.1 バージョン管理システムを用いたソフトウェア開発

近年，バージョン管理システムを用いたソフトウェア開発が広く行われている．バージョン管理システムを用いたソフトウェア開発では，第 2 章で述べたように複数の開発者が協調かつ並行して開発を行う．このような開発では，複数の開発者が並行して行った作業を，互いに独立してバージョン管理システムのリポジトリに反映させることが頻繁に行われる．このとき，複数の開発者が同じファイルを変更していた場合，変更が競合してしまい，先に反映された変更内容が後から反映された変更内容で上書きされてしまう場合がある．これに対して，競合の解消を助ける研究が行われている [24]．しかし，全ての競合に適用することはできず，現在でも競合の解決は労力のかかる作業である．また，変更によってバグが発生した場合，そのまま変更内容が反映されるとソフトウェアに問題が発生する可能性がある．

こうした問題を避けるため，バージョン管理システムの運用において，図 4 のように管理者を置く場合がある．管理者を置くことにより，管理者という第三者が変更内容を確認することで，バグがリポジトリに反映されることを避けることができる．管理者を置いた場合のソフトウェア開発は次のように行われる．

1. 開発者はリポジトリから最新の状態のソフトウェアを取得する．

2. 開発者はソフトウェアを変更する．

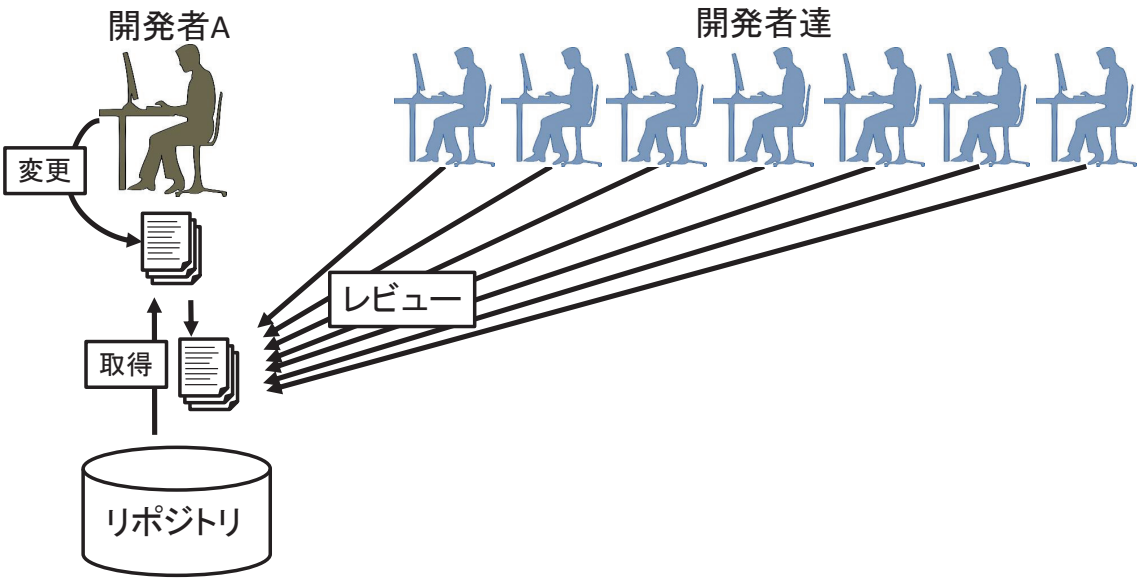


図 5: コードレビュー

3. 変更が完了した開発者は，管理者に変更内容の確認を依頼する．

4. 依頼を受けた管理者は，変更内容を確認する．

5. 変更内容にバグや競合などの問題が無ければ，管理者は変更内容をリポジトリに反映する．

6. 変更内容に問題があった場合は，変更内容をリポジトリに反映しない．

GitHub[25] では，変更内容の確認を機能として取り入れており，GitHub から派生した様々なサービスにおいても同様の機能が実現されている．このような手順でソフトウェア開発を行うことにより，ソフトウェアの品質の低下を防ぎ，問題の発生により労力が余分にかかる可能性を削減する．しかし，こうしたソフトウェア開発において管理者は，様々な開発者から送られてくる大量の依頼を受けて変更内容を確認する必要がある．開発を効率的に行うために，管理者が変更内容を理解するために必要な労力は少ないほどよい．

また，バージョン管理システムを用いたソフトウェア開発において，図 5 のようにコードレビューが行われることがある [26][27]．コードレビューを行うことにより，ソフトウェア開発において大きく 3 つの利点が挙げられる．

- 管理者を置いた場合と同様に第三者が変更内容を確認することにより，バグがリポジトリに反映される可能性が減少する．

- 開発者が自身の行った変更に対してフィードバックを受けることができるため，変更を行った開発者のスキルが向上する．
- 他人が書いたコードを精密に確認しレビューを行うことで，変更を行った開発者以外のコードレビューを行う開発者達のスキルも向上する．

以上の利点から，コードレビューはソフトウェア開発において広く行われている．しかし，コードレビューを行う開発者達は他人が変更したソースコードを理解する必要がある．開発を効率的に行うためには，コードレビューに要する労力は少ない方が望ましい．

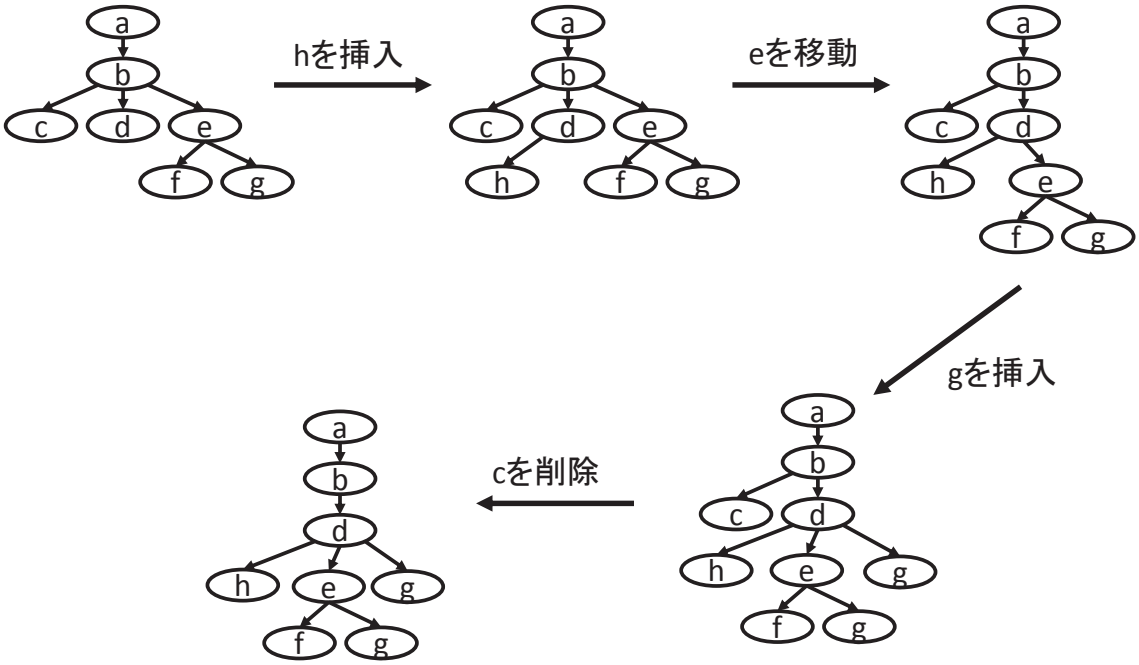
こうした理由から，変更内容が開発者にとって理解しやすく表現されることにより，ソフトウェア開発にかかる労力を削減することができる．これまでにも，2 つのバージョンにおけるソースファイルの変更内容をテキストに基づいて表現する手法が広く研究されている [5][6][7]．例えば，Unix の diff コマンドでは，2 つのファイルが与えられたとき，一方のファイルからもう一方のファイルへの変更における，行単位での追加や削除を表現することができる [5]．テキスト行粒度で差分を生成するアルゴリズムは，非常に早く，また，プログラミング言語から独立して変更内容を表現する．

しかし，これらのアルゴリズムは，行未満の細かい粒度での変更内容を表現することができない．また，プログラミング言語とは独立して差分を生成するため，プログラムの構造を考慮して変更内容を表現することができない．さらに，diff においては追加と削除しか考慮せず，更新や移動といったソフトウェア開発において頻繁に行われる操作を表現することができない．これらの問題点により，テキストに基づく変更内容の表現が，開発者にとって理解しやすい表現でない場合がある．

こうした問題を解決するため，抽象構文木に基づく変更内容の表現方法がこれまでに数多く研究されている．抽象構文木に基づいて表現することにより，プログラムの構造を考慮して変更内容を表現することができる．例えば，新しく挿入されたノードが関数の宣言を意味する場合，新しい関数がコード内に追加されたことを表現することができる．こうした特徴から，抽象構文木に基づく表現はテキストに基づく表現よりも開発者にとって変更内容をわかりやすく表現することができる．抽象構文木に基づく表現は，与えられた 2 つのソースファイルから Edit Script を生成することによって，変更内容を表現する．

3.2 Edit Script

任意の 2 つの AST が与えられたとき，一方の AST をもう一方の AST に変更する手順は Edit Script を用いて表すことができる [9][11][12]．Edit Script は複数の操作から構成される．一般に，Edit Script を構成する操作とそれらの Edit Script 上での表現は以下の 4 種類である．



(a) AST の操作

```
Insert(h, d, 1, …, …)
Move(e, d, 2)
Insert(g, d, 3, …, …)
Delete(c)
```

(b) 対応する Edit Script

図 6: Edit Script の例

挿入 : $Insert(t, t_p, i, l, v)$

ノードの追加を表す．挿入するノードの ID として t，ノードのラベルとして l，ノードの値として v，挿入後に親ノードとするノードの ID として t_p，何番目の子にするかを表す数値として i を引数に持つ．

削除 : $Delete(t)$

ノードの削除を表す．削除するノードの ID として t を引数に持つ．

更新 : $Update(t, v)$

ノードの値の更新を表す．更新の対象となるノードの ID として t，ノードに格納する新しい値として v を引数に持つ．

移動 : $Move(t, t_p, i)$

部分木の移動を表す．移動する部分木の根ノードの ID として t，移動先の親ノードの

ID として t_p，何番目の子にするかを表す数値として i を引数に持つ．

図 6a に AST への編集の例を示す．図 6a では，AST において，ノード h をノード d の 1 番目の子として挿入し，ノード e をノード d の 2 番目の子として移動，ノード g をノード d の 3 番目の子として挿入，ノード c を削除，の順で操作を行っている．このように，AST の編集は対象の AST に操作を適用することで行う．また，この場合の Edit Script は図 6b である．ただし，ここでは挿入されたノードのラベルと値は省略している．

Edit Script を構成する操作の個数を **Edit Script の長さ**と呼ぶ．Edit Script の長さが長いほど，多くの操作が含まれる．多くの操作が行われる変更は開発者にとって理解に要する労力が大きい．よって，Edit Script の長さは開発者が変更内容を理解するために要する労力を表す．そのため，Edit Script はその長さが短いほど，開発者にとって理解しやすい Edit Script であるといえる．

3.2.1 既存の Edit Script 生成手法

Edit Script を生成するためにこれまでに多くの手法が研究されている [8][9][10][11]．Edit Script を生成する手法のうち，AST におけるノードの追加，削除，更新を考慮する手法が既に広く研究されている [28]．追加，削除，更新を考慮する手法は多く存在するが，その中で最も短い時間で Edit Script を生成する手法は RTED[10] である．しかし RTED は計算に $O(n^3)$ かかるため，大きなソースファイルに対して実用的な時間の範囲内で Edit Script を生成することができない．また，これらの手法は，ソフトウェア開発において頻繁に現れる，ノードの移動を識別することができず，移動元のノードを全て削除し移動後のノードを全て追加するという方法で変更内容を表現することになる．その結果，Edit Script は長くなり，理解しづらいものとなる．

ノードの移動を考慮して短い Edit Script を生成することは NP 困難である．しかし，実用的な経験則を用いることにより，移動を考慮した上で短い Edit Script を生成する手法が研究されている．その内で最も有名なものが Chawathe らのアルゴリズム [11] であり，移動を含む Edit Script を効率的に生成することができる．しかし，Chawathe らのアルゴリズムには制約があり，ソースファイルを表現した細粒度な AST にそのままでは適用することができない．

また，XML 文書を対象として Edit Script を生成するアルゴリズムが提案されている [29][30]．これらのアルゴリズムは Chawathe らのアルゴリズムと異なり制約を持たない．しかし，これらのアルゴリズムは速度を最も重視しており，開発者にとって変更内容を理解しやすい Edit Script を生成することを重視していない．

AST において Edit Script を生成するアルゴリズムのうち，最も有名なアルゴリズムが

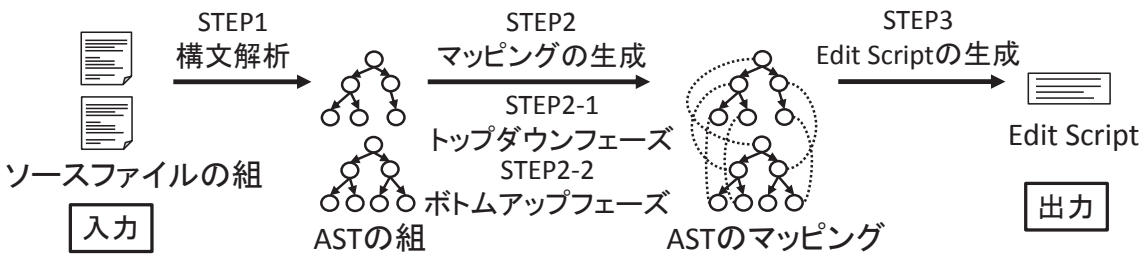


図 7: GumTree の概要

ChangeDistiller[9] である．ChangeDistiller は Chawathe らのアルゴリズムに影響を受けており，AST においてより効果的に動作するように調整されている．しかし，ChangeDistiller は簡略化した AST を対象としており，1 つの文に多くの要素を持ちうる言語において，ChangeDistiller は細粒度な Edit Script を生成できない．

3.2.2 GumTree

Falleri らは，細粒度な AST において，実用的な時間の範囲内で移動を考慮した短い Edit Script を生成する GumTree を開発した [12]．GumTree の概要を図 7 に示す．GumTree は Falleri らが考案した GumTree アルゴリズムにより，操作として移動を含んだ Edit Script を生成し，実用的な実行時間で動作することを可能としている [12]．GumTree は図 8a のようなソースファイルの組を入力として与えると，図 8b のような形式で Edit Script を生成し出力する．

GumTree による処理を解説する．GumTree は入力として与えられたソースファイルの組に対して下記の 3 つの STEP を適用することで Edit Script を生成し出力する．

STEP1: 構文解析

与えられたソースファイルの組のそれぞれについて構文解析を行い，ソースファイルの組に対応する AST の組を生成する．

STEP2: マッピング

STEP1 で生成した AST の組について，下記の 2 つの STEP を経て 2 つの AST における対応するノードや部分木の組についてマッピングを生成する．

STEP2-1: トップダウンフェーズ

AST の組において，AST に基づく Type-2 コードクローンの検出を行うことで，それぞれの AST に共通して存在する部分木を発見する．

変更前

```
public class Test{
 public String foo(int i){
   if(i == 0) return "Foo!";
 }
}
```

変更後

```
public class Test{
 private String foo(int i){
   if(i == 0) return “Bar";
   else if(i == -1) return “Foo!”;
 }
}
```

(a) 入力例

insert(t_1, a, 2, ReturnStatement, ε)
insert(t_2, t_1, 1, StringLitteral, Bar)
insert(t_3, a, 3, IfStatement, ε)
insert(t_4, t_3, 1, InfixExpression, ==)
insert(t_5, t_4, 1, SimpleName, i)
insert(t_6, t_4, 2, PrefixExpression, -)
insert(t_7, t_6, 1, NumberLiterral, 1)
move(b, t_3, 2)
update(c, private)

(b) 出力例

図 8: GumTree の入出力例

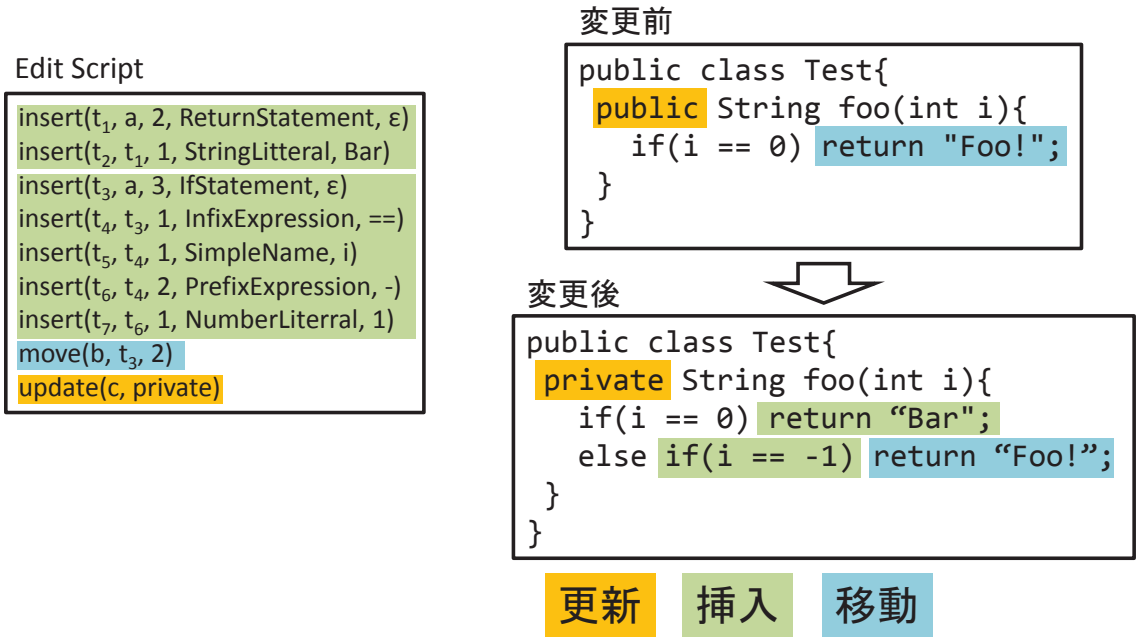


図 9: GumTree による Edit Script に基づいた変更内容の表示

STEP2-2: ボトムアップフェーズ

STEP2-1 において発見した部分木のマッピングを基準として，ここまででマッピングに含まれていない部分木のうち類似する部分木の組を発見する．処理の一部において，一定以下の大きさの AST に対して RTED によるマッピング作成を利用している [10].

STEP3: Edit Script 生成

STEP1 で生成した 2 つの AST と，STEP2 で生成したマッピングを用いて Edit Sciript を生成する．Edit Script の生成には Chawathe らのアルゴリズムを用いる [11].

また，GumTree は生成した Edit Script に基づき，変更内容をソースコード上に表すことができる．Edit Script に基づく変更内容の表示例を図 9 に示す．図 9 の左側が GumTree が生成した Edit Script であり，その Edit Script に基づいた表示を右側に示している．緑色でハイライトした部分が挿入に対応する．図中ではソースコードの 2 箇所への挿入が行われている．黄色でハイライトした部分が更新を表している．青色でハイライトした部分が移動を表しており，図中では return 文が if 文の直後から if 文中の else 節中に移動している．

3.3 研究動機

第 3 章で述べたように，ソフトウェア開発において，コピーアンドペーストが頻繁に行われることがわかっている．コピーアンドペーストを含む変更を GumTree に与えて Edit

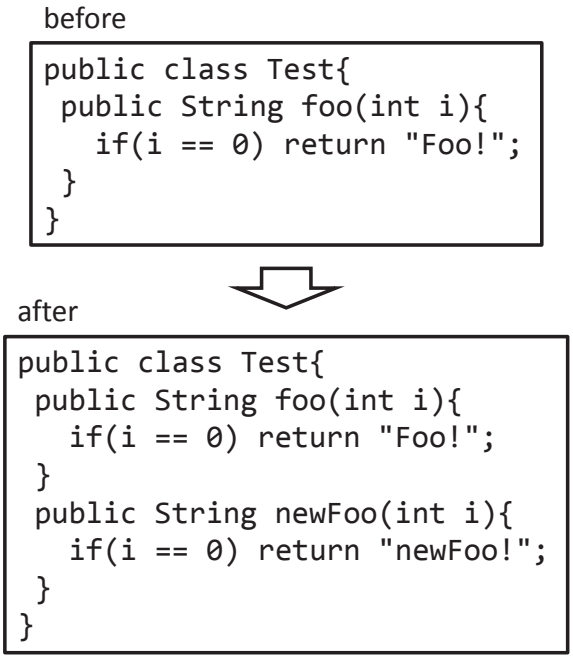


図 10: コピーアンドペーストが含まれる変更

Script を生成する場合，Edit Script は挿入，削除，更新，移動の組み合わせで表されるため，新しいノードが多数挿入されたと表される．

図 10 のようにコピーアンドペーストを含む変更を GumTree に与えた場合を例として説明する．after において，新しいメソッド newFoo が追加されている．メソッド newFoo はメソッド Foo と Type-2 クローンの関係にある．このとき，図 11 のように GumTree は before と after をそれぞれ AST に変換する．図中の AST がそれぞれ before と after のソースコードに対応している．また，説明のため AST の各ノードの ID としてアルファベットを付加した．この変更に対して，GumTree が生成した Edit Script は図 12 である．Edit Script より，この変更において様々な新しいノードが 15 個挿入されたことがわかる．

しかし，実際に挿入されたノードは新しいノードではなく，既存のノードと同じノードである．そのため，既存のノードを利用することにより，この Edit Script をより簡潔に表すことができる．また，この例において，挿入された部分木は既存の部分木と同じ形である．よって，部分木の根ノードのみへの操作として表すことで，Edit Script を大幅に短縮できる．

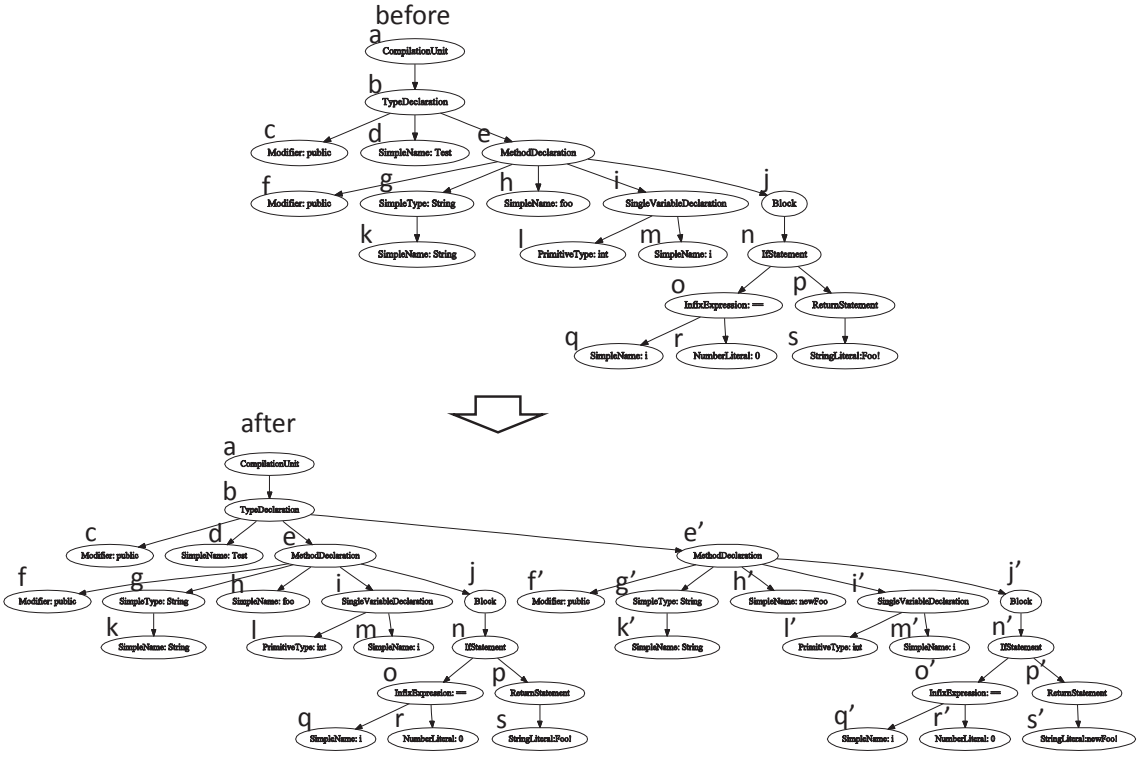


図 11: コピーアンドペーストが含まれる変更を AST に変換

```
Insert(e', b,  4, MethodDeclaration, ε)
Insert(f', e', 1, Modifier, public)
Insert(g', e', 2, SimpleType, String)
Insert(k', g', 1, SimpleName, String)
Insert(h', e', 3, SimpleName, newFoo)
Insert(i', e', 4, SingleVariableDeclaration, ε)
Insert(l', i', 1, PrimitiveType, int)
Insert(m', i', 2, SimpleName, i)
Insert(j', e', 5, Block, ε)
Insert(n', j', 1, IfStatement, ε)
Insert(o', n', 1, InfixExpression, ==)
Insert(q', o', 1, SimpleName, ε)
Insert(r', o', 2, NumberLiteral, 0)
Insert(p', n', 2, ReturnStatement, ε)
Insert(s', p', 1, StringLiteral, newFoo!)
```

図 12: コピーアンドペーストが含まれる変更における GumTree による Edit Script

```
C&P(e, b, 4)
Update(h', newFoo)
Update(s', "newFoo!")
```

図 13: コピーアンドペーストを含む場合の提案手法による Edit Script

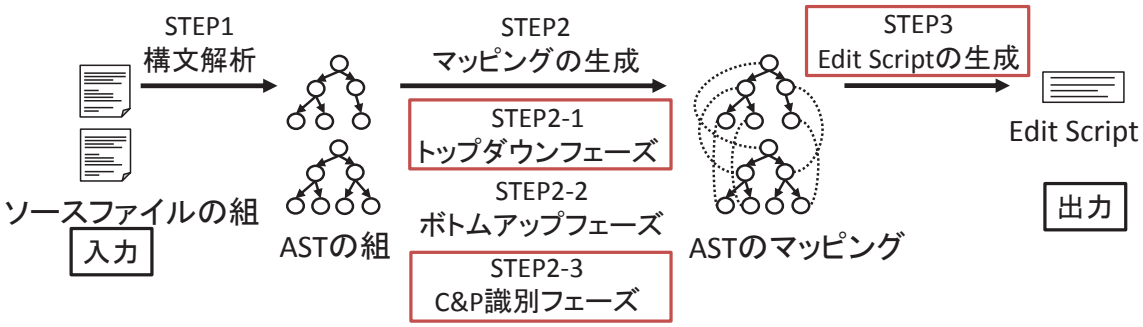


図 14: 提案手法の概要

4 提案手法

4.1 概要

第 3.3 節で説明したように，コピーアンドペーストされた部分木を挿入の繰り返しではなく 1 つのコピーアンドペーストとして表すことで，開発者にとってより理解しやすい Edit Script を生成できる．そのため，提案手法では Edit Script を構成する操作として，挿入，削除，更新，移動に加えてコピーアンドペーストを追加することで，より開発者にとって理解しやすい Edit Script を生成する．生成する Edit Script において，コピーアンドペーストは既存の部分木と同じ部分木を別の場所にも追加するという 1 つの操作として表すことができる．例えば，第 3.3 節の例のように図 10 を提案手法に与えると，生成される Edit Script は図 13 のようになる．この例において，提案手法が生成する Edit Script は GumTree が生成する Edit Script よりも短いだけでなく，コピーアンドペーストや更新といった具体的な変更内容をイメージしやすい．そのため，提案手法が生成する Edit Script は GumTree が生成する Edit Script よりも開発者にとって理解しやすいといえる．

本研究では GumTree を拡張することで提案手法を設計している [12]．提案手法の概要を図 14 に示す．提案手法はソースファイルの組を入力とし，5 つの STEP を経て生成した Edit Script を出力する．3 章で示したように，コピーアンドペーストを考慮することにより，開発者がソースファイルへの変更の内容をより理解しやすい Edit Script を生成できる．提案手法において，Edit Script を構成する操作は挿入，削除，更新，移動，コピーアンドペー

ストの 5 種類であり，Edit Script において以下のように表現される．

挿入 : $Insert(t, t_p, i, l, v)$

ノードの追加を表す．挿入するノードの ID として t，ノードのラベルとして l，ノードの値として v，挿入後に親ノードとするノードの ID として t_p，何番目の子にするかを表す数値として i を引数に持つ．

削除 : $Delete(t)$

ノードの削除を表す．削除するノードの ID として t を引数に持つ．

更新 : $Update(t, v)$

ノードの値の更新を表す．更新の対象となるノードの ID として t，ノードに格納する新しい値として v を引数に持つ．

移動 : $Move(t, t_p, i)$

部分木の移動を表す．移動する部分木の根ノードの ID として t，移動先の親ノードの ID として t_p，何番目の子にするかを表す数値として i を引数に持つ．

コピーアンドペースト : $C\&P(t, t_p, i)$

部分木のコピーアンドペーストを表す．コピーアンドペーストする部分木の根ノードの ID，コピーアンドペースト先の親ノードの ID，何番目の子にするかを表す数値を引数に持つ．

4.2 手順

提案手法は GumTree のアルゴリズムを拡張している．図 14 では，提案手法において拡張あるいは追加した STEP を赤い実線で囲んだ．

STEP1: 構文解析

与えられたソースファイルの組のそれぞれについて構文解析を行い，ソースファイルの組に対応する AST の組を生成する．

STEP2: マッピング

STEP1 で生成した AST の組について，下記の 3 つの STEP を経て 2 つの AST における対応するノードや部分木の組についてマッピングを生成する．

STEP2-1: トップダウンフェーズ

AST の組において，Type-2 のコードクローン検出を行うことで，それぞれの AST に共通して存在する部分木を発見する．さらに，平行してコピーアンドペーストによって作成された部分木の候補の記録も行う．

STEP2-2: ボトムアップフェーズ

STEP2-1 において発見した部分木のマッピングを基準として，ここまででマッピングに含まれていない部分木のうち類似する部分木の組を発見する．

STEP2-3: C&P 識別フェーズ

STEP-2-1 において発見した，コピーアンドペーストによって作成された部分木の候補である，2 つの AST に共通して存在する部分木の組のうち，STEP2-1 と STEP2-2 においてマッピングされていないものを発見し，コピーアンドペーストによって作成された部分木とする．

STEP3: Edit Script 生成

STEP1 で生成した 2 つの AST と，STEP2 で生成したマッピングを用いて Edit Sciript を生成する．STEP2-3 において識別した部分木をコピーアンドペーストによって作成するものとして処理する．

以降，提案手法において拡張あるいは追加した STEP について詳細に説明する．

4.3 STEP2-1: トップダウンフェーズ

本 STEP では GumTree において行われる処理と並行してコピーアンドペーストの候補を探す処理を行う．変更前のソースファイルから生成された AST と，変更後のソースファイルから生成された AST を比較することにより，マッピングを行う．本 STEP は下記の手順で実行される．

1. Type-1 クローンの関係にある部分木を発見し記録する．並行して Type-2 クローンの関係にある部分木も記録する．
2. Type-1 クローンの関係にある部分木のうち，1 対 1 の関係にある組をマッピングに追加する．
3. Type-1 クローンの関係にある部分木のうち，1 対多あるいは多対多の関係にある部分木について，最も類似度が高い組のみをマッピングに追加する．ここで用いる類似度は GumTree において定義された類似度と同じである．
4. ここまでで残った Type-1 クローンの関係にある部分木の組と，Type-2 クローンの関係にある部分木の組をコピーアンドペーストの候補として記録する

4.4 STEP2-3: C&P 識別フェーズ

本 STEP は GumTree には存在せず，提案手法で追加した STEP である．本 STEP では下記の手順により C&P の識別を行う．

1. 変更後のソースファイルから生成された AST を先行順に探索し，下記の条件を満たすノードを探す．
   - ここまでの STEP においてマッピングに追加されていない
   - コピーアンドペーストの候補としてトップダウンフェーズで記録されている
2. 条件を満たすノードを発見すると，変更前のソースファイルから生成された AST において対応するノードとともに，コピーアンドペーストされたノードとしてマッピングに追加する．
3. コピーアンドペーストされたとしてマッピングに追加されたノードを根とする部分木に含まれる全ての子ノードをコピーアンドペーストの候補から除外する．

また，コピーアンドペーストの識別において制約を設けた．提案手法では，Type-2 のクローン検出により共通する部分木を発見するため，識別子名やリテラルなどは正規化される．そのため，文法上の理由により，偶然同じ形になっただけの部分木もコピーアンドペーストとして識別される可能性がある．例えば，変数宣言において，宣言する変数の変数名と，代入する値の双方が異なっていても，部分木は同じ形となるため，コピーアンドペーストと識別される可能性がある．このような誤検出を防ぐため，提案手法では，部分木の形が同じであっても，その部分木を構成するノードが持つ全ての値が異なる場合はコピーアンドペーストとして識別しない．

4.5 STEP3: Edit Script 生成

Edit Script の生成において，挿入，削除，移動，更新の検出は GumTree と同様に Chawathe らのアルゴリズムを用いる [11]．ただし，提案手法では，Chawathe らのアルゴリズムを拡張して，検出する操作としてコピーアンドペーストを加えている．本 STEP では AST を順に 2 回探索することにより Edit Script を生成している．

1. 変更後のソースファイルを解析し生成した AST を幅優先探索することにより挿入，移動，更新に加えてコピーアンドペーストを検出する
2. 変更前のソースファイルを解析し生成した AST を後行順に探索することにより削除を検出する

幅優先探索の際に，C&P 識別フェーズにおいてコピーアンドペーストとしてマッピングされたノードを発見すると，そのノードはコピーアンドペーストされたノードであると判断し，Edit Script にコピーアンドペーストを追加する．

5　実験

5.1　準備

本実験では，提案手法と GumTree の比較を行った．第 3 章で述べたように，Edit Script を生成する手法として GumTree 以外の手法も存在する．しかし，Falleri らの研究において，GumTree がそれらの手法より，変更内容を開発者にとって理解しやすく表現できることが示されている [12]．そのため，提案手法と GumTree を比較することで，提案手法の有効性を評価する．比較のため，提案手法と GumTree をツールとして下記のように実装した．

提案手法: 追加，削除，更新，移動，コピーアンドペーストから構成される Edit Script を生成する．

GumTree: 追加，削除，更新，移動から構成される Edit Script を生成する．

提案手法と GumTree において，下記のしきい値を用いた．

- 不要なマッピングを減らすため，トップダウンフェーズにおいてマッピングする部分木の高さの最小値を 2 とする．
- ボトムアップフェーズにおいて部分木をマッピングする際に用いる類似度の最小値を 0.5 とする．
- 計算時間を短縮するため，ボトムアップフェーズにおいてマッピングする部分木のノード数の最大値を 100 とする．

これらのしきい値は Falleri らの実験において設定された値である [12]．

本実験では，実験対象として 14 のプロジェクトで構成されるデータセットである CVS-Vintage dataset を用いた [31]．CVS-Vintage dataset には 43,250 個のソースファイル，ソースファイルへの変更が行われた 352,182 個のリビジョンが含まれる．

5.2　手順

本実験では，2 つのツールを用いて Edit Scritp を生成し，生成した Edit Script を比較することで評価を行う．本実験では，あるソースファイルが変更されたとき，変更される前のソースファイルと，変更された後のソースファイルの間には 1 つの変更が行われたものとして扱う．1 つの変更には複数の操作が含まれうる．また，変更前のソースファイルと変更後のソースファイルを 2 つのソースファイルとみなし，合わせて変更と呼ぶ．

ツールへの入力として与える変更は CVS-Vintage dataset に含まれる 14 個のプロジェクトから取得する．変更は各プロジェクトのリポジトリのそれぞれから取得する．図 15 に，リ

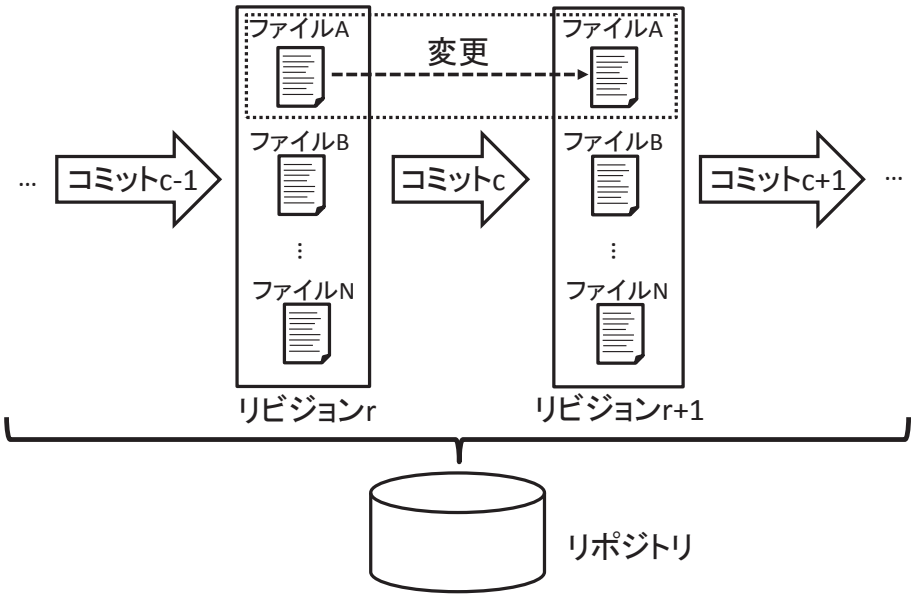


図 15: リポジトリからの変更の取得

ポジトリ内における変更の例を示す．この例では，コミット c においてファイル A に変更が行われ，それによりリビジョン r+1 が作成されている．実験では各プロジェクトから，それぞれ 1000 個までの変更を取得する．変更が 1000 個未満しか存在しない場合には，そのプロジェクトのリポジトリに含まれる変更を全て使用する．実験に用いる変更の取得は下記の手順で行う．

1. 各リポジトリから，ソースファイルが変更されたリビジョン（図 15 におけるリビジョン r+1 に対応する）を取得し，そのリビジョンにおいて変更されたソースファイルをチェックアウトする．

2. 取得したソースファイルのそれぞれについて，直前のリビジョン（リビジョン r に対応する）から対応するソースファイルをチェックアウトする．

3. 対応する変更前のソースファイルと変更後のソースファイルを 1 つの変更として保存する．

4. ソースファイルが変更された全てのリビジョンにおいて同様に変更を取得する

5. 取得した変更から，提案手法に入力として与えた際に長さ 1 以上の Edit Script を生成する変更のみを残し，それ以外の変更を取り除く．

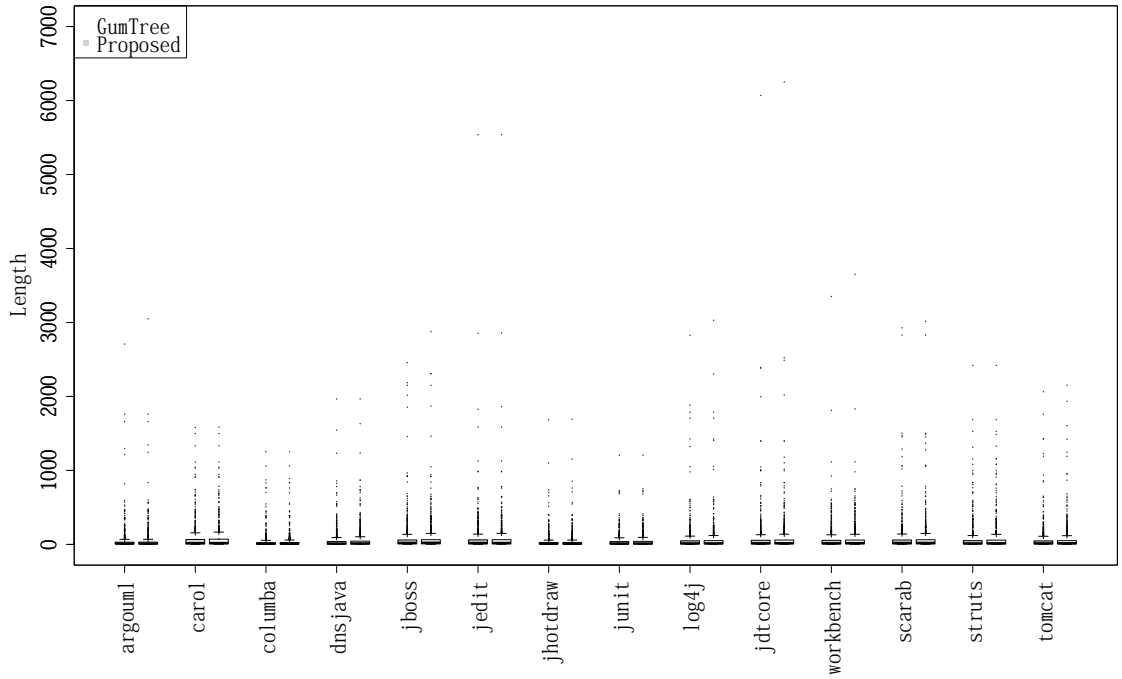


図 16: 各ツールによる Edit Script の長さ

6. 提案手法によって長さ 1 以上の Edit Script が生成される変更からランダムに 1000 個までを取得する．

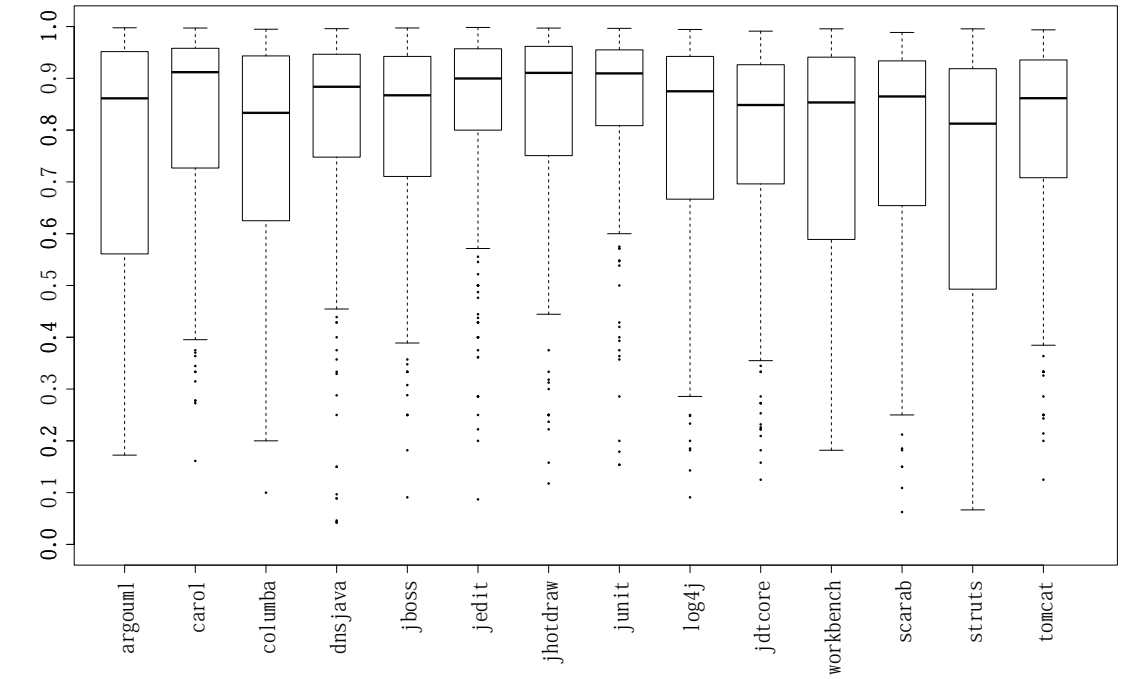
以上の手順から，13699 個の変更を取得した．こうして取得した変更を入力とし，提案手法と GumTree へ入力として与え，Edit Script の長さと実行時間を取得しする．

5.3 評価基準

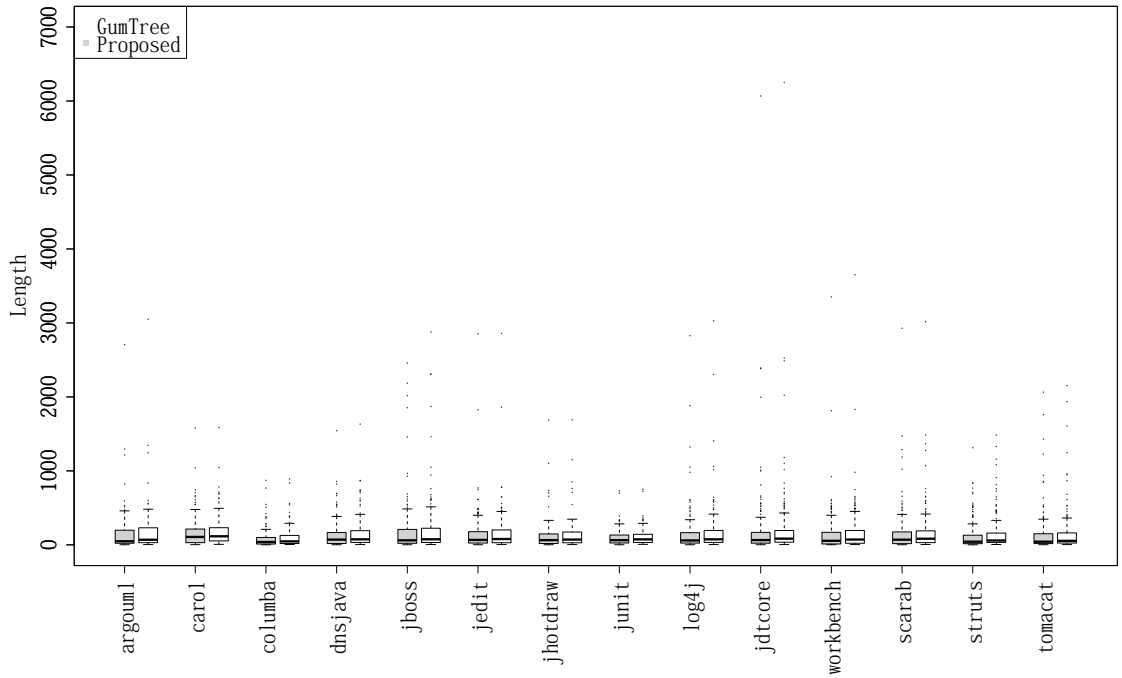
本実験では，各ツールが生成する Edit Script の長さと各ツールが Edit Script を生成するために要する実行時間を評価基準とした．Edit Script が短いほど，開発者が変更の内容を理解するための負担が少なくなる．そのため，より短い Edit Script を生成するツールほど優れている．

5.4 結果

提案手法と GumTree が生成する Edit Script の長さを図 16 に箱ひげ図を用いて示す．縦軸が Edit Script の長さを表す．横軸が対応するソフトウェアを表す．ウィルコクソンの符号

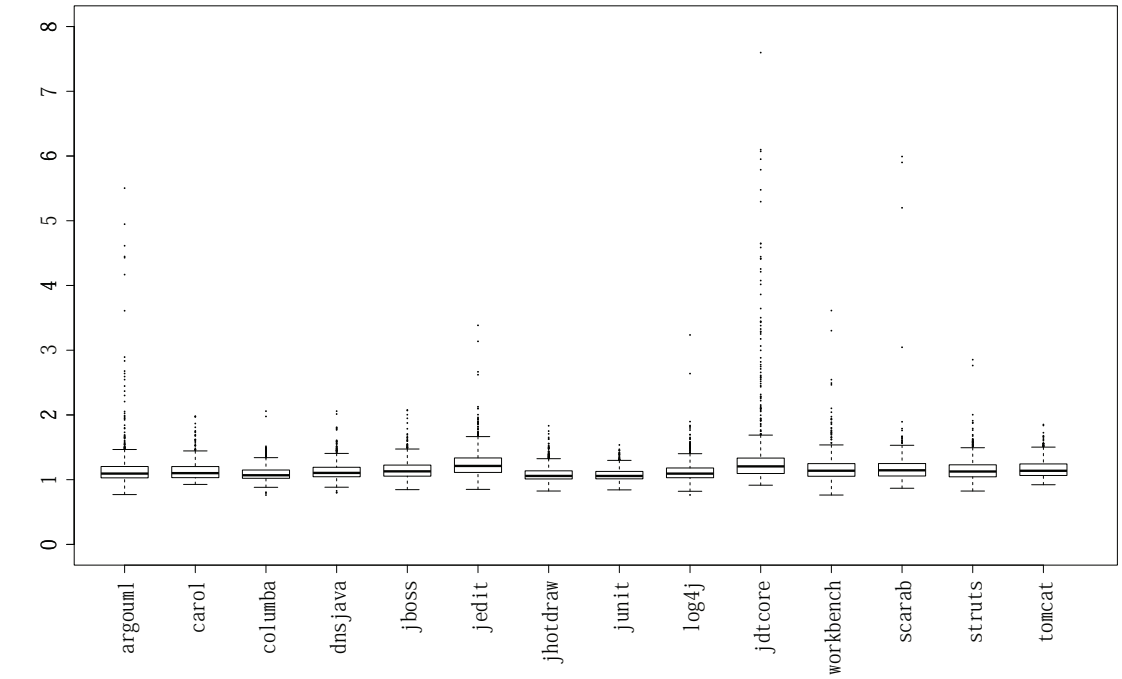


(a) 各ツールによる Edit Script の長さの比較

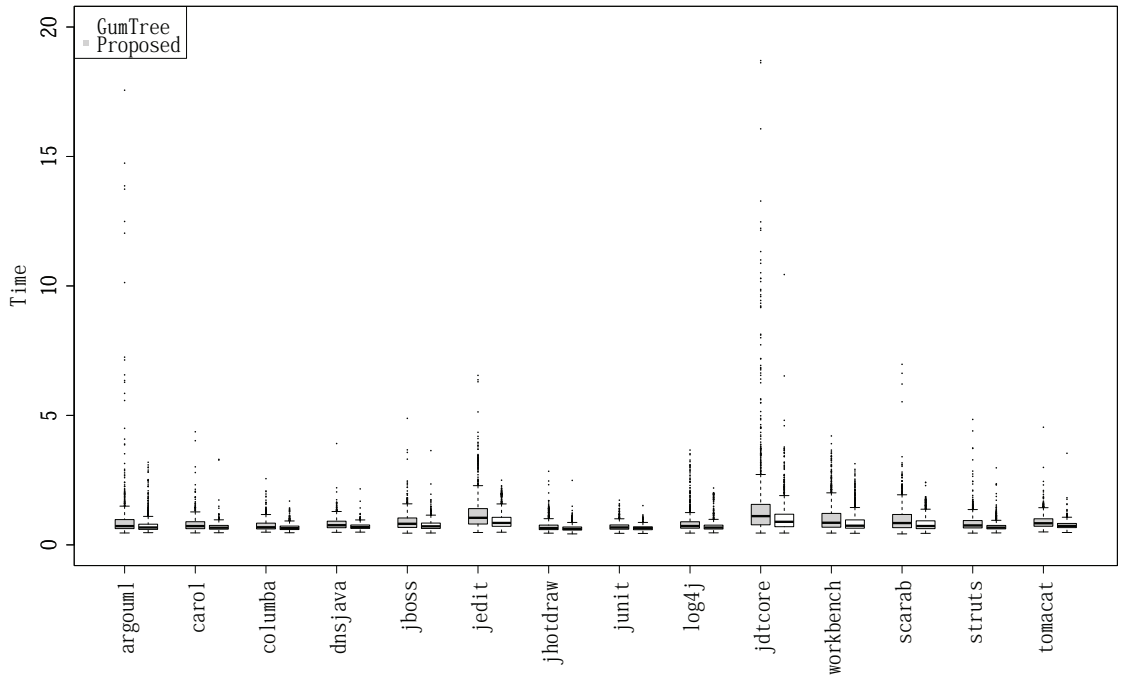


(b) 各ツールによる Edit Script の長さ

図 17: 各ツールによって Edit Script の長さが異なる場合



(a) 各ツールの実行時間の比較



(b) 各ツールの実行時間

図 18: 全ての変更における実行時間

提案手法が生成する Edit Script が短い	GumTree が生成する Edit Script が短い	等しい
2,491(18%)	0(0%)	11,208(82%)

表 1: Edit Script の長さの比較

	最大値		第 3 四分位数		中央値		第 1 四分位数		最小値	
	GumTree	提案手法	GumTree	提案手法	GumTree	提案手法	GumTree	提案手法	GumTree	提案手法
argouml	3,049	2,708	231	198	69	49	30	22	4	2
carol	1,586	1,581	230.5	214	116	107	53.75	32	7	3
columba	892	872	128	100	47	39	18	11	5	1
dnsjava	1,632	1,544	192	166.5	75	71	33	18	4	2
jboss	2,876	2,457	223.75	208.5	75.5	61.5	32	21.25	5	2
jedit	2,858	2,853	202	178	79	67	29	24.5	4	2
jhotdraw	1,691	1,686	172.75	144	71.5	64	28.75	18.75	4	2
junit	751	728	141.75	133	74.5	63.5	32.75	25.75	7	2
log4j	3,029	2,827	195	165	75	60	33	24	4	2
jdtcore	13,269	12,628	194.5	168.5	84	65	36	22.5	4	2
workbench	3,651	3,351	195	170	72	55	22	12	4	2
scarab	3,016	2,928	189.5	175.5	83	70	30.5	16.5	5	2
struts	1,486	1,313	159.25	129.25	59	43	33	16	5	1
tomcat	2,152	2,064	162	150.5	54	43	24.5	17.5	4	2

表 2: ツールによって Edit Script の長さが異なる場合における Edit Script の長さ

順位検定を用いて有意水準 α =0.05 で各ツールが生成する Edit Script の長さを検定したところ，各ツール間で統計的に有意な差があることがわかった．また，提案手法と GumTree が生成する Edit Script の長さの比較結果を表 1 に示す．表 1 から，18%の変更において提案手法が生成する Edit Script の方が短く，GumTree が生成する Edit Script の方が短い場合は存在しなかった．82%の変更では提案手法と GumTree において Edit Script が同じ長さであった．以降，用いるツールによって生成される Edit Script の長さが異なる変更について調査した．

表 2 において，各ツールにおける Edit Script の長さの四分位数を示した．各ソフトウェアが表中の行に対応しており，四分位数が列に対応している．各列中において GumTree と提案手法での長さを左右に並べて示している．表 2 より，全ソフトウェアの全四分位数において，提案手法の方が短い．また，各ツールによる Edit Script の長さを図 17b に箱ひげ図を用いて示した．縦軸が Edit Script の長さを表し，横軸が対応するソフトウェアを表す．どのソフトウェアにおいても，提案手法の方が短い．

図 17a では，箱ひげ図を用いて，各変更において提案手法を用いることにより Edit Script

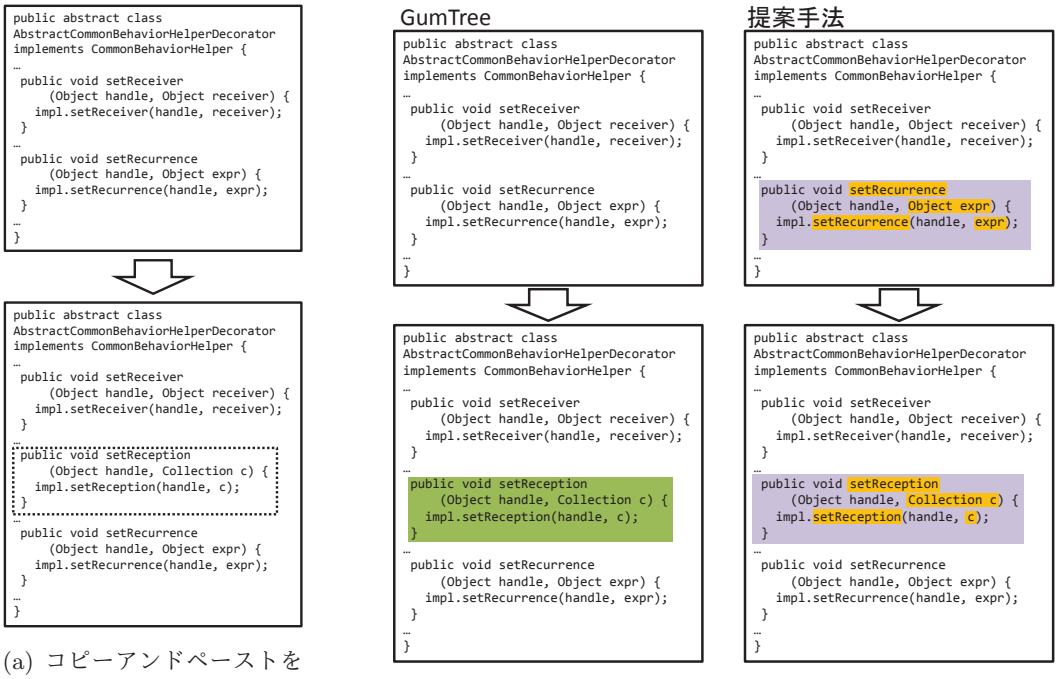
がどの程度変化するかを表した．縦軸が GumTree が生成する Edit Script の長さを 1.0 としたときの提案手法が生成する Edit Script の長さを表す．横軸が対応するソフトウェアを表す．図 17a より，どのソフトウェアにおいても 1.0 以下となっているため，提案手法と GumTree によって生成される Edit Script の長さが異なる場合には，必ず提案手法が生成する Edit Script の方が短い．58.5%の場合において Edit Script が 10%以上短縮された．また，図 17b に，Edit Script の長さを箱ひげ図で表した．どのソフトウェアにおいても提案手法において Edit Script が短くなっていることがわかる．

図 18b に，全ての変更における，提案手法と GumTree の実行時間を箱ひげ図で示した．縦軸が実行時間を表す．単位は秒である．横軸が対応するソフトウェアを表す．ウィルコクソンの符号順位検定を用いて有意水準 α =0.05 で各ツールの実行時間を長さを検定したところ，各ツール間で統計的に有意な差があることがわかった．また，図 18a では，箱ひげ図を用いて，各変更において提案手法を用いることにより実行時間がどの程度変化するかを表した．縦軸は，GumTree を用いた場合の実行時間を 1 としたときの提案手法の実行時間を表す．横軸は対応するソフトウェアを表す．GumTree と比較して提案手法は実行時間が長くなっている．ただし，96%の場合において提案手法は GumTree の 1.5 倍以下の実行時間で Edit Script を生成することができる．また，75%の場合において提案手法は 1 秒以内に Edit Script を生成し，96%の場合において 2 秒以内に Edit Script を生成することができる．

5.5 考察

5.5.1 提案手法によって Edit Script を大幅に短縮できる場合

実験において，Edit Script を大幅に短縮できた場合の具体例を説明する．図 19a に ArgoUML において行われた変更を示す．図上部のソースコードが変更前のソースコードであり，図下部のソースコードが変更後のソースコードである．矢印は変更が行われたことを示している．この変更では，変更前にあったメソッドを再利用し，図中の点線で囲まれた setReception というメソッドを追加している．setReception の内部では，変数名やメソッド呼び出し文で呼び出すメソッドの名前は異なるが非常に似た処理を行っている．この変更について，提案手法と GumTree の双方による変更内容の表現は図 19b のようになる．左側が GumTree による表現であり，右側が提案手法による表現である．図上部の変更前のソースコードにおいて，紫色でハイライトされたコード片はコピーアンドペーストされたコード片であり，このコード片が別の箇所にも追加されることがわかる．黄色でハイライトされたコード片は更新される前のコード片であり，このコード片が別のコード片に更新されることがわかる．図下部の変更後のソースコードにおいて，緑色でハイライトされたコード片は新たに挿入されたコード片である．黄色でハイライトされたコード片は更新された後のコード



(a) コピーアンドペーストを含む変更の例

(b) GumTree と提案手法による表現の比較

図 19: 提案手法によって Edit Script を短縮できる場合

片であり，更新の結果このようなコード片になったことがわかる．紫色でハイライトされたコード片はコピーアンドペーストによって作成されたコード片であり，変更前のソースコード中の対応するコードがコピーアンドペーストされてこのコードが作成されたことがわかる．

GumTree による表現では，変更後のソースコードにおいて，setReception という新しいメソッドが挿入されたことがわかる．setReception がどのように実装されているかはソースコードを読まなければわからない．提案手法による表現では，変更前のソースコードにおける setRecurrence を再利用して変更後のソースコードにおける setReception が作成されたことがひと目でわかり，コピーアンドペースト後にどこが更新されたかもわかる．また，非常によく似たメソッドということがわかるため，ソースコードを読む上でも理解しやすい．この例では，明らかに提案手法が，開発者にとってより理解しやすい変更内容の表現を行っている．この例における Edit Script の長さは，GumTree は 20，提案手法は 5 である．つまり，提案手法を用いることにより Edit Script は 75%短縮されている．

5.5.2 Edit Script の長さが等しい場合

実験において，提案手法と GumTree が生成する Edit Script の長さが等しい場合が数多く見つかった．この場合において，Edit Script にコピーアンドペーストを含む場合と，Edit Script にコピーアンドペーストを含まない場合を計測した．計測した結果を表 3 に示した．

コピーアンドペーストを含む	コピーアンドペーストを含まない
0	11,208

表 3: Edit Script の長さが等しい場合

コピーアンドペーストの後に，コピーアンドペーストされた部分木の全てのノードにおいて更新が行われた場合，提案手法による Edit Script は 1 だけ長くなるはずである．また，コピーアンドペーストの後に，コピーアンドペーストされた部分木のうち 1 つを除く全てのノードにおいて更新が行われた場合，提案手法による Edit Script の長さは GumTree と等しくなるはずである．しかし，表 3 より，提案手法と GumTree による Edit Script の長さが等しいとき，コピーアンドペーストは含まれないことがわかった．また，表 1 より，提案手法による Edit Script の方が GumTree による Edit Script より長い場合は存在しない．つまり，コピーアンドペーストが含まれる場合には必ず提案手法によって Edit Script は短縮されていた．これは，提案手法を実装する際にコピーアンドペースト検出において設けた制約によるものと考えられる．また，実際には部分木を構成するノードの全てが値を持つことは稀である．

5.6 妥当性への脅威

実験では Java にて記述されたソースファイルのみを使用した．提案手法や GumTree のアルゴリズムは Java の特徴とは無関係だが，他の言語を対象とした場合において異なる結果が得られる可能性がある．

実験において提案手法と GumTree の双方において，しきい値は Falleri らの研究において用いられた値と同じ値を使用している．しきい値が異なれば，各ツールは異なる振る舞いをする．実験結果について，しきい値の影響を評価するにはより多くの実験が必要とされる。

実験において，Falleri らの実験と同様に，実験対象として CVS-Vintage dataset を使用した．他のデータセットを実験対象に使用した場合に異なる結果が得られる可能性がある．

6　おわりに

本研究では，ソースファイルの変更内容を開発者にとって理解しやすく表現するために，コピーアンドペーストを考慮することにより Falleri らの手法を改良した．また，提案手法の有効性を評価するために評価実験を行った．実験では，提案手法と Falleri らの手法を比較することで提案手法を評価した．実験において，CVS-Vintage dataset を用いて 14 個のソフトウェアの開発履歴から 13,699 個の変更を実験対象として取得した．取得した変更に対して提案手法と Falleri らの手法がそれぞれ生成する Edit Script の長さと，Edit Script の生成に要する実行時間を比較した．その結果，提案手法を用いることにより，Falleri らの手法より短い Edit Script を生成できることを確認した．また，ほとんどの変更に対して提案手法は実用的な時間で Edit Script を生成できることを確認した．

今後の課題は以下の通りである．

- より多くのソフトウェアに対して実験を行う．
- 被験者実験を行い，コピーアンドペーストを考慮した表現により，開発者にとって変更内容がどの程度理解しやすくなったか調査する．
- Type-3 クローンをコピーアンドペーストとして識別できるよう手法を改良する．

謝辞

本研究を行うにあたり，理解あるご指導を賜り，常に励まして頂き，いざというときにも心強いお力添えを頂きました 楠本 真二 教授に心より感謝申し上げます．

本研究を行うにあたり，熱心なご指導を頂き，有益かつ的確なご助言を頂きました 肥後 芳樹 准教授に心より感謝申し上げます．

短い期間ではありましたが，本研究において多くのご意見を頂き，また，日々の研究室生活を盛り上げて頂きました 柗本 真佑 助教 に心より感謝申し上げます．

本研究を行うにあたりご指導，ご協力を頂き，さらに日常でも声をかけて頂き，様々な問題に対して深い理解のもと助言を頂きました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士後期課程 3 年の 村上 寛明 氏，同 楊 嘉晨 氏に深く感謝申し上げます．

本研究や研究室生活において，常に切磋琢磨し高め合い，ときに励まし合ってきました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 2 年の 江川 翔太 氏，同 高 良多朗 氏に心より感謝申し上げます．

本研究を行うにあたり，多大なるご助言，ご助力を頂きました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 1 年の 横山 晴樹 氏に心より感謝申し上げます．

本研究を行うにあたり，様々な場面で心強いお力添えを頂きました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 1 年の 小倉 直徒 氏，同 佐飛 祐介 氏，同 鷲見 創一 氏，同 幸 佑亮 氏に心より感謝申し上げます．

本研究を行うにあたり，日常の議論の中でご助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 1 年の 古田 雄基 氏に心より感謝申し上げます．

本研究や日々の研究室生活など，様々な場面において親切なご協力を頂きました，大阪大学基礎工学部情報科学研究科 4 年の 下仲健斗 氏，同 中島弘貴 氏，同 山田悠斗 氏，同 山本将弘 氏に心より感謝申し上げます．

また，本研究に関して多くのご助言を頂くとともに，様々な面において親切なご助力，ご協力を頂きました楠本研究室の皆様に心より感謝致します．

最後に，本研究に至るまでに，講義，演習，実験等でお世話になりました大阪大学大学院情報科学研究科の諸先生方に，この場を借りて心から御礼申し上げます．

参考文献

[1] T. Mens and S. Demeyer. *Software Evolution*. Springer, 1 edition, 2008.

[2] CVS - Concurrent Versions System. `http://www.nongnu.org/cvs/`.

[3] Apache subversion. `http://subversion.apache.org/`.

[4] Git. `http://git-scm.com`.

[5] E. W. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, Vol. 1, pp. 251–266, 1986.

[6] W. Miller and E. W. Myers. A file comparison program. *Software: Practice and Experience*, Vol. 15, No. 11, pp. 1025–1040, 1985.

[7] M. Asaduzzaman, C. K. Roy, K. A. Schneider, and M. D. Penta. Lhdiff: A language-independent hybrid approach for tracking source code lines. In *ICSM*, pp. 230–239. IEEE Computer Society, 2013.

[8] M. Hashimoto and A. Mori. Diff/ts: A tool for fine-grained structural change analysis. In *WCRE*, pp. 279–288, 2008.

[9] B. Fluri, M. Wuersch, M. PInzger, and H. Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Trans. Softw. Eng.*, Vol. 33, No. 11, pp. 725–743, November 2007.

[10] M. Pawlik and N. Augsten. Rted: A robust algorithm for the tree edit distance. *Proc. VLDB Endow.*, Vol. 5, No. 4, pp. 334–345, December 2011.

[11] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. *SIGMOD Rec.*, Vol. 25, No. 2, pp. 493–504, June 1996.

[12] J. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pp. 313–324, 2014.

[13] M. Kim, L. Bergman, T. Lau, and D. Notkin. An ethnographic study of copy and paste programming practices in oopl. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pp. 83–92, 2004.

[14] T. M. Ahmed, W. Shang, and A. E. Hassan. An empirical study of the copy and paste behavior during development. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pp. 99–110, 2015.

[15] M. M. Lehman. On understanding laws, evolution, and conservation in the large-program life cycle. *J. Syst. Softw.*, Vol. 1, pp. 213–221, September 1984.

[16] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multi-linguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654–670, July 2002.

[17] I. Baxter, A. Yahin, M. Anna L. Moura, and L. Bier. Clone detection using abstract syntax trees. *Proc. of International Conference on Software Maintenance 98*, pp. 368–377, Mar. 1998.

[18] S. Uchida, A. Monden, N. Ohsugi, T. Kamiya, K. Matsumoto, and H. Kudo. Software analysis by code clones in open source software. *Journal of Computer Information Systems*, Vol. XLV, No. 3, pp. 1–11, Apr. 2005.

[19] 井上克郎, 神谷年洋, 楠本真二. コードクローン検出法. コンピュータソフトウェア, Vol. 18, No. 5, pp. 47–54, 2001.

[20] 肥後芳樹, 楠本真二, 井上克郎. コードクローン検出とその関連技術. 電子情報通信学会論文誌, Vol. J91-D, No. 6, pp. 1465–1481, 2008.

[21] 神谷年洋, 肥後芳樹, 吉田則裕. コードクローン検出技術の展開. コンピュータソフトウェア, Vol. 28, No. 3, pp. 28–42, 8 2011.

[22] S. Bellon. Detection of software clones. *Technical Report, Institute for Software Technology, University of Stuttugart*, 2003. available at http://www.bauhaus-stuttgart.de/clones/.

[23] S. Bellon, R. Koschke, G. Antniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *IEEE Transactions Software Engineering*, Vol. 31, No. 10, pp. 804–818, Oct. 2007.

[24] 早瀬康裕, 松下誠, 井上克郎. 構文木の差分を用いた版管理システム向きマージ機能. 情報処理学会論文誌, 2007.

[25] GitHub. `https://github.com/`.

[26] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, Vol. 11, No. 3, pp. 309–346, July 2002.

[27] J. Asundi and R. Jayant. Patch review processes in open source software development communities: A comparative case study. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, HICSS '07, pp. 166c–, 2007.

[28] P. Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, Vol. 337, No. 1-3, pp. 217–239, June 2005.

[29] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml documents. In *Inproceedings of the 18th International Conference on Data Engineering*, pp. 41–52, 2002.

[30] R. Al-Ekram, A. Adma, and O. Baysal. diffx: An algorithm to detect changes in multi-version xml documents. In *Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research*, pp. 1–11, 2005.

[31] M. Monperrus and M. Martinez. Cvs-vintage: A dataset of 14 cvs repositories of java software. Technical Report hal-00769121, INRIA, 2012.