特別研究報告

題目

再利用が存在するリポジトリ間におけるファイルの起源検出

指導教員
井上 克郎 教授

報告者
川満 直弘

平成 26 年 2 月 14 日

大阪大学 基礎工学部 情報科学科

平成 25 年度 特別研究報告

再利用が存在するリポジトリ間におけるファイルの起源検出

川満 直弘

内容梗概

ソフトウェアの開発において，再利用は頻繁に行われている．バグや脆弱性が再利用元のプロジェクトにおいて発見，修正された場合，開発者は新しいバージョンのファイルを再び取り込み直す必要がある．ファイルの更新を継続的に行っていくためには，開発者が再利用したファイルがどのライブラリ，どのバージョンに由来するかを把握している必要がある．過去の調査により，多くのプロジェクトにおいて，利用しているオープンソースのライブラリが脆弱性を抱えているバージョンのままであるか，使用しているライブラリのバージョンに関する情報が管理されておらず失われた状態になっていることが判明している．再利用に関する情報が欠落した理由の 1 つは，それが自動的に記録される情報ではないことである．

本研究では，与えられた 2 つのリポジトリに対し，一方のリポジトリに含まれたファイルが，他方のリポジトリのどのリビジョンのファイルを起源としているか，それに加えリポジトリ内で行われた修正をどの程度取り込んだか (組成) を検出する手法を提案する．ケーススタディの結果，1 つのケースに対しては，ファイルの対応および再利用の方向はすべて正しく，リビジョンの対応が正しく検出できたのは 8 割程度であった．また，別のケースに対しても半分程度のファイルについて起源を正しく特定できた．組成情報に関しては，変更の部分的な取り込みの値が意図せず高くなってしまう場合が多いという結果になり，今後の改善が必要とされる．

主な用語

起源解析
再利用
バージョン管理システム

目 次

1　はじめに

ソフトウェアの開発において，再利用は頻繁に行われている．再利用の形式としては，バイナリをそのまま用いたり，コピーしたコードに何らかの変更を加えて利用したりといった様々な方式があるが，大規模開発にも広く使われている C 言語の場合，必要な機能を持つライブラリのソースファイルを直接プロジェクトに取り込むことがしばしば行われている．再利用の活用は，有用なソフトウェアを短期間で開発するために必要不可欠である．しかし，再利用を行うことで，ファイルに含まれたバグや脆弱性を取り込んでしまう恐れもある．そういった不具合が再利用元のプロジェクトにおいて発見，修正された場合，開発者は新しいバージョンのファイルを再び取り込み直す必要がある．

ファイルの更新を継続的に行っていくためには，開発者が再利用したファイルがどのライブラリ，どのバージョンに由来するかを把握している必要がある．しかし，ライブラリの再利用状況についての調査 [4] によると，多くのプロジェクトにおいて，利用しているオープンソースのライブラリが脆弱性を抱えているバージョンのままであるか，使用しているライブラリのバージョンに関する情報が管理されておらず失われた状態になっていることが判明した．重大なセキュリティに関する脆弱性の告知などではバージョン番号が指定されている [10] が，バージョン番号が失われてしまうと開発者は対応の必要性の有無すら判断することができない．

再利用に関する情報が欠落した理由の 1 つは，それが自動的に記録される情報ではないことである．ソフトウェア開発のプロジェクトにおいて，プロジェクト内部のファイルの更新は，Git[9] などのバージョン管理システムを用いてリポジトリと呼ばれるデータベースに記録されている．開発者が変更を行い，リポジトリに登録する都度，リビジョン番号が自動的に割り当てられ，開発者はリビジョン番号の指定によって，任意の時点のファイルの情報を自由に参照することができる．一方，リポジトリ間で行った再利用は，リポジトリには自動的には記録されない．したがって，ファイルやコミットメッセージなどの形で開発者が意図的に残さなければ，そういった情報は失われてしまう．

本研究では，与えられた 2 つのリポジトリに対し，一方のリポジトリに含まれたファイルが，他方のリポジトリのどのリビジョンのファイルを**起源**としているか，それに加えリポジトリ内で行われた修正をどの程度取り込んだか (**組成**) を検出する手法を提案する．これにより，開発者が再利用したコードの出所を知ることができ，脆弱性の有無など再利用後に生じた問題を容易に確認することを可能とする．また，開発者が再利用元リポジトリでのファイルの更新を取り込むことも容易になる．具体的な実現方法としては，ファイルのリビジョン間の類似度の計算を使用している．一方のリポジトリの各リビジョンと，他方のリポジトリの各リビジョンを比較し，類似度が高いものを再利用の候補として抽出する．そして，

類似度がある閾値より高かった最も古いリビジョンの組をファイルの再利用のタイミングとして検出し，先に存在したリビジョンを再利用元，後から作られたリビジョンを再利用先とみなす．再利用先のファイルが再利用元のファイルと完全には一致しない場合，再利用元のファイルに加えられている更新のうち，どれだけの割合が再利用先のファイルに加えられている可能性があるか，組成を計算する．

以降，第2章では研究の背景と関連する研究について述べる．第3章では提案手法について説明し，第4章ではケーススタディとしてリポジトリに手法を適応し，手法の有効性について考察する．最後に第5章でまとめと今後の課題について述べる．

2 背景

2.1 コードの再利用

ソフトウェア開発において，再利用を伴った開発が広く行われており，特にオープンソースソフトウェア (OSS) の再利用が頻繁に行われている．OSS の大きな特徴は，制限されず無料で再頒布されたものが利用できること，ソースコードが含まれていること等が挙げられる [7].

OSS を製品開発に取り込む利点として，以下のようなものが挙げられている [1].

- 市販のサプライヤーの製品は，突然製品の開発やサポートを終了させる恐れがあるのに対し，デファクトスタンダードとなっていて大規模なユーザーコミュニティを持っている OSS の場合，長期間利用できると期待される．
- コンポーネントの更新，修正のサイクルを短縮できる．
  - ローカライズや不具合の修正を無料で受けることができる．
  - 組み込みシステムにおいては，高速で，新しいドライバーやハードウェア関連の機能を提供する．
- プロプライエタリで独自のインターフェイスを持つ製品を使用している場合，契約条件の変化などで製品を変更するコストが非常に高い．OSS の API やバイナリ互換性などのデファクトスタンダード上でアプリケーションを開発することで，その危険を回避できる．
- ソフトウェアハウスは，OSS を利用してコストを削減し，多数のソフトウェアコンポーネントに依存するというリスクを減少させる．
- OSS はセキュリティが向上している．ソースコードが公開されているため，プロプライエタリなソフトウェアに比べて多くのユーザが使用し，ソースコードを閲覧する．そのため，多くの不具合は，早期に修正され，広く通知される．

2.2 脆弱性を抱える可能性のあるコードの再利用

コードの再利用は有用であるが，一方で再利用したコードに含まれるバグやセキュリティ上の脆弱性をプロジェクトに取り込んでしまう危険性もある．脆弱性を抱えるバージョンの OSS を再利用している例は，Xia らの調査によって報告されている [4]．Xia らはオープンソースのライブラリのコードを再利用しているプロジェクトについて，使用しているライブラリのバージョンが脆弱性を抱える可能性のあるバージョンであるかどうかを調査した．そ

の結果，調査対象としたプロジェクトのうち，zlib を利用しているプロジェクトでは 31.1%，libcurl を利用しているプロジェクトでは 85.7%，libpng を利用しているプロジェクトでは 92%のプロジェクトが，脆弱性を抱える可能性のあるバージョンを利用していることが判明した．脆弱性を抱える可能性のあるバージョンのライブラリを使用することは，アプリケーションのクラッシュを引き起こす場合や，任意のコードが実行される恐れがあるなどの問題を孕んでおり，不具合が修正されたバージョンへ更新することが推奨される．

2.3 バージョン管理システム Git

ソフトウェア開発において，ファイルのバージョンはバージョン管理システムによって管理されている．バージョン管理システムには，CVS, Subversion, Git 等が存在する．本研究では，Git[9] を対象にする．

バージョン管理システムとは，ある時点でのファイルの情報を記録し，後になってそのファイルや情報を取り出すことができるようにするシステムである．これにより，開発者がファイルの内容を誤って変更したり，ファイルを削除したりした場合にも，リポジトリのファイルを取り出すことで被害を軽減することができる．ファイルの情報が記録されたものをリポジトリと呼ぶ．開発者は，作業しているファイルに変更を加え，その変更の内容を保存するために，リポジトリに記録を行う．このリポジトリに記録することをコミットすると言い，コミットする単位はコミット，もしくはリビジョンと呼ばれる．記録される情報には，ファイルの内容だけでなく，コミットに対応するチェックサム，作者の名前とメールアドレス，コミット日時，コミットメッセージなどの情報が記録されている．

開発者が常にリポジトリの最新のコミットに変更を加えてその内容をコミットすることを繰り返せば，履歴は一直線になる．しかし，2 つ以上のコミットがある 1 つのコミットを元にしている場合がある．例えば，開発と保守などの別々の目的がある場合などである．このとき，ブランチと呼ばれる機能を使うことで，開発者は履歴を分岐させる．また，分岐していた履歴を統合して 1 つのコミットとすることも可能であり，この行為をマージと呼ぶ．このように，リポジトリの履歴に枝分かれや合流が含まれるため，Git の履歴は閉路を持たない有効グラフの形で表される．

バージョン管理システムに記録される情報は，単一のプロジェクトに関するものである．再利用元と再利用先は通常別個のリポジトリで管理されており，リポジトリ間での再利用が行われた際の記録はどちらのリポジトリにも自動的に記録されない．再利用元のどのリビジョンのファイルを取り込んだのか，その後の再利用元の更新に追随しているかという情報は，開発者がコミットメッセージなどに記録しない限り失われてしまう．

2.4 起源分析

本研究は，ファイルの起源情報を取り出す起源分析の一種である．起源分析は Godfrey によって提唱された考え方である [2]．ソースコードが変更や統合，分割を繰り返すことにより開発者の元々の意図や変更理由が失われてしまう．そのため，ソースコードの起源を解析するための技術が提案されている．

ファイル単位で他のソフトウェアプロジェクトから類似するファイルを検出する手法としては，Inoue らの Ichi Tracker[3] が挙げられる．Ichi Tracker では，コード断片をソースコード検索エンジンで検索し，検索結果のソースファイルをコード断片との類似度でクラスタリングする．その上で，ソースファイルを時系列順に並べることによりソースコードの再利用の経緯を可視化する．Inoue らの手法では，起源を知りたいコード断片を入力として与えるのに対し，本手法ではリポジトリ間の比較を行うことで，どのファイルが再利用されたのかという点からソースファイルの起源を解析する．

プロジェクト間で同一内容のファイルを検出する手法としては，佐々木らの FCFinder がある [14]．これは，大規模なソフトウェア群中のソースコードの等しいファイルを高速に検出可能な手法である．本研究では，対象をソフトウェアのリポジトリとし，リビジョン間の関係の抽出を目指している．

Kanda らは，ソフトウェアプロダクトの集合を入力として元になったプロダクトを推定する手法を提案した [6]．この手法の出力はプロダクト間の派生関係であり，ファイル単位での詳細な関係性は手作業で確認する必要がある．

3　提案手法

本研究では，与えられた 2 つのリポジトリに対し，一方のリポジトリに含まれたファイルが，他方のリポジトリのどのリビジョンのファイルを**起源**としているか，それに加えリポジトリ内で行われた修正をどの程度取り込んだか (**組成**) を検出する手法を提案する．例えば，与えられたリポジトリの組 A，B に対して，リポジトリ A のファイル x.c のリビジョン 1.2 はリポジトリ B のファイル x.c のリビジョン 2.1 を元にして，リビジョン 2.3 から 2.4 で施された修正を加えたものである，というように再利用ファイルの組の情報を列挙する．これにより，開発者が再利用したコードの出所を知ることができ，脆弱性の有無など再利用後に生じた問題を容易に確認することを可能とする．また，開発者が再利用元リポジトリでのファイルの更新を取り込むことも可能になる．

以降では，あるファイルの再利用元としたファイルのリビジョンの情報を起源情報，起源情報に加えそのファイルに施された修正の情報を組成情報と呼ぶ．提案手法は，バージョン管理システムやプログラミング言語に依存しないが，本研究では Git で管理された 2 つのリポジトリで管理されている C 言語のソースを対象として実装を行った．

提案手法は，以下の手順で構成される．

step 1 各リポジトリに格納されたそれぞれのファイルに対し，ファイルグラフを構築する．ファイルグラフとは，履歴中で 1 つのファイルパスが 1 つのファイルに対応すると仮定し，1 つのファイルの履歴を有効グラフとして表したものである．ファイルグラフのノードは 1 つのリビジョンに対応する．

step 2 リポジトリ間において再利用されたと推定されるファイルグラフを組として検出する．

step 3 検出されたファイルグラフの組に対し，どちらが再利用元/先かを推定する．

step 4 再利用先と推定されたファイルグラフ中のリビジョンのファイルについて，再利用元のファイルグラフ中のから再利用元となったリビジョンのファイルを検出する．

step 5 再利用先と推定された各リビジョンのファイルについて，変更の部分的な取り込みの割合を求める．変更の部分的な取り込みとは，再利用元のファイルのリビジョン間の変更を，部分的に再利用先のファイルに加えることである．

3.1　ファイルグラフの構築

リポジトリに記録されている全てのファイルを対象に，以下の手順を行う．図 1 に，これらの手順の適用例を示す．図 1 の対象リポジトリには，リビジョン 1 の時点でファイル B,C

が含まれており，ファイルBの内容が変更されてリビジョン2となっている．その後，ファイル A,C の内容に変更が加えられ，ファイル B が削除されたものがリビジョン 3 となり，ファイル C が削除され，ファイル A の名前が B へと変更されたことでリビジョン 4 となった履歴が格納されている．

1. 1 つのファイルがリポジトリ内に存在する期間，すなわちファイルが追加されてから，削除されるコミットもしくは履歴の上で子を持たないコミットまでの期間の履歴を 1 つの有向グラフで表す．ノードはリビジョンに対応し，対応するコミット履歴上で親に当たるノードが親，子に当たるノードが子となるようにエッジを追加する．同じファイルパスであっても，1 度削除された後に再び加えられるような場合は，別のグラフになる．

2. Git によりファイルパスが変更されたことが検知できれば，変更前のファイルに対応するグラフの変更直前にあたるノードと，変更後のファイルに対応するグラフの変更直後のノードの間にエッジを加え，それらのグラフを連結して 1 つのグラフとする．

3. それぞれのグラフについて，対象とするファイルに変更が加えられなかったノードを取り除く．その際に親であったノードから子であったノードへのエッジを加える．

4. あるファイル F に対するファイルグラフは，ファイル F をノードとして含む有効グラフをすべて取り出し，1 つのグラフとしたものとする．例えば，図 1 においてファイル A に対応するファイルグラフはグラフ 1 そのものである．ファイル B に対応するファイルグラフはグラフ 1 とグラフ 2 からなる非連結グラフである．ファイル C に対応するファイルグラフはグラフ 3 そのものである．

3.2 再利用が行われたファイルを表すファイルグラフの組の検出

片方のリポジトリに含まれるあるファイルが，もう片方のリポジトリに含まれるあるファイルを再利用していると推定される場合，そのファイルに対応するファイルグラフの組を検出する．

片方のリポジトリのファイルグラフに含まれる各リビジョンのファイルと，もう片方のリポジトリのファイルグラフに含まれる各リビジョンのファイル間について，以下の手順で類似度を求める．

1. 比較するリビジョンのファイルの組それぞれに対し，トークン分割を行う．トークン分割とは，ソースコードの字句解析を行い，トークン列へ変換することである．ここで，空白空行，コメントは除去する．

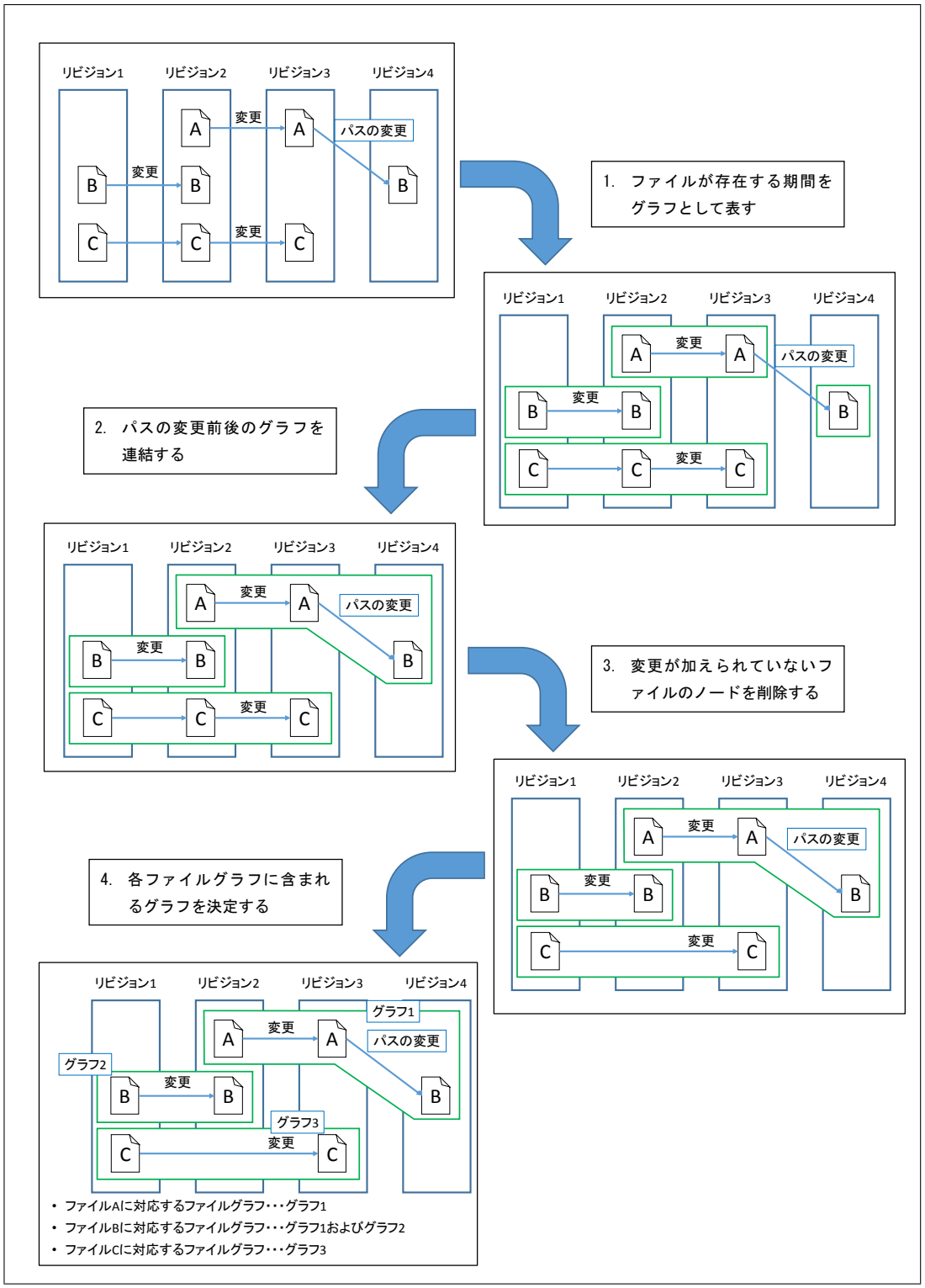

図 1: ファイルグラフの構築過程

2. 得られた 2 つのトークン列 A,B に対して，類似度 $sim(A, B)$ を以下のように定義する

$$sim(A, B) = \frac{|A| + |B| - |diff|}{|A| + |B| + |diff|}$$

ここで，$|A|$ および $|B|$ は列 A，B の要素数を表す．また，$|diff|$ は列 A から列 B へ，要素の挿入，削除のみで変形させることができる最小要素数を表す値であり，以下の関係がある．

$$|diff| = |A| + |B| - |LCS| \times 2$$

ここで，$|LCS|$ は列 A,B 間の Longest Common Subsequence(最長共通部分列) の長さを表す．

求められた類似度が閾値 th 以上であるようなリビジョンの組が 1 つでもあれば，ファイルグラフの間で再利用が行われたと判断する．

3.3 再利用元/先の検出

3.2 において求めたファイルグラフの組について，どちらのファイルグラフからどちらのファイルグラフに再利用されたかを推定する．

片方のリポジトリのファイルグラフに含まれる各リビジョンのファイルと，もう片方のリポジトリのファイルグラフに含まれる各リビジョンのファイル間について，3.2 節の方法で類似度を求める．双方のファイルグラフについて，あるリビジョンのファイルがもう片方のファイルグラフに含まれるリビジョンのファイルのうちのいずれかとの類似度が閾値を越える物の集合を求める．求めた 2 集合の最もコミット日時の古いものについて，それらのファイルについてコミット日時を比較し，日時の古いファイルグラフを再利用元，新しいファイルグラフを再利用先とする．

3.4 再利用先ファイルグラフに対応する再利用元ファイルグラフの検出

再利用先となったファイルグラフに含まれるそれぞれのリビジョンのファイルに対し，再利用元のファイルグラフ中から最も類似するリビジョンのファイルを，再利用元のリビジョンのファイルとして検出する．

1. コピー先の対象のリビジョンのファイルと，コピー元のファイルグラフ中の各リビジョンのファイルの間で，トークン分割を行って要素とした列を入力とした類似度を求め，最高値と等しいリビジョンのファイルからなる集合を求める．トークン分割の際は，コメントは除去する．

2. コピー先の対象のリビジョンのファイルと，上記で求めた集合中の各リビジョンのファイルの間で，1 行を 1 要素とした列を入力として類似度を求め，最高値と等しいリビジョンのファイルからなる集合を求める．行末の改行文字は，要素には含まない．

3. 上記で求めた集合のうち，コミットに記録された日時の最も古いものをコピー元のリビジョンのファイルとする．

3.5 変更の部分的な取り込みの割合の検出

再利用元として検出されたリビジョンのファイルと，再利用先として検出されたリビジョンのファイルの間の差分が，再利用元リポジトリ内で加えられた差分を含んでいるかを出力する．ここで，差分とはある 2 ファイルに対し，一方のファイルからもう一方のファイルに変形する際に必要になる，削除が必要な行および追加が必要な行のことである．また，それぞれの行に対して追加の行なのか削除の行なのかという情報を含む．

再利用が検出された各リビジョンのファイルに対し，以下の手順を行う．

1. 再利用元とされたファイルグラフ中において，全ての親子のノードに対応するリビジョンのファイル間の差分を求める．

2. 再利用元として検出されたリビジョンのファイルと，再利用先として検出されたリビジョンのファイルの間の差分を求める．

3. 1 により得た各差分が，2 により得られた差分にどの程度含まれているかを出力とする．比較は行単位とする．2 の差分に含まれる 1 の行数を分子，1 の行数を分母とした割合としてあらわされる．

$$(\text{変更の部分的な取り込みの割合}) = \frac{(2\text{ の差分に含まれる }1\text{ の行数})}{(1\text{ の行数})}$$

4　ケーススタディ

起源情報が正しく検出できるかどうかを確認するため，2 件の再利用が行われたリポジトリの組に対し手法を適用してその出力を評価した．また，部分的な変更の取り込みが実際に行われている例を調査し，手法によりその取り込みを出力できるかどうかを調査した．

今回のケーススタディでは，類似度の閾値を 0.8 とした．

再利用元リビジョンを正しく検出できるかどうかの評価のため，項目の一つとして到達性を定義する．これは，手法がファイルの内容を元に，ファイル単位で再利用元リビジョンを求めることに由来する．複数のコミット間においてファイルの内容が変更されなかった場合，同じ内容のファイルが複数のリビジョンに存在することになり，実際に再利用を行ったリビジョンを特定することはできないからである．検出されたコミットのノードを A，期待されるコミットのノードを B とする．以下の条件のいずれかに当てはまる場合，到達性があると定義する．

条件 1　コミット A とコミット B が同じものである

条件 2　コミット A がコミット B の祖先であり，コミット A からコミット B へのあるパス上，ただしコミット A を含まずコミット B を含む範囲において，該当ファイルの変更を含むコミットのノードが存在しない

この定義により，到達性のあるコミット同士は，該当ファイルの内容が同一である．

4.1　fs2open および libpng

fs2open は，ゲームエンジンのプロジェクトである [8]．また，libpng は，PNG フォーマット用のグラフィックライブラリである [11]．

fs2open のリポジトリのログに，libpng を取り込んだという記述が 3 件見つかった．図 2 は，そのコミットのログである．

手法により，延べ 57 件のファイルの起源情報を検出した．再利用の方向は，57 件のすべてにおいて再利用元を libpng，再利用先を fs2open と検出した．

リポジトリのログから，fs2open が libpng のファイルを取り込んだ際に，libpng プロジェクトのファイルがディレクトリ構造を変更せず fs2open の libpng ディレクトリ以下にコピーされたことがわかった．ここでは，再利用先のパスが，再利用元リポジトリのパスの先頭に libpng/を加えたものである場合，ファイルの対応が正しく取れていると判断する．例えば，再利用元のパスが png.c だった場合，再利用先のパスが libpng/png.c だった場合に正しく，dir/dir/png.c といった再利用先のディレクトリが異なる場合や，libpng/file.c というようにファイル名が異なる場合はファイルの対応が正しく取れていないとする．

```
commit 58f9e77468cbf0c525d5f4a109f94372873ee984
Author: Zacam <Zacam@387891d4-d844-0410-90c0-e4c51a9137d3>
Date:   Tue Jan 10 21:43:30 2012 +0000
    Fix for Mantis 0002558 : Resolves the issue by updating the libpng provided
to version 1.5.7.

    git-svn-id: svn://svn.icculus.org/fs2open/trunk/fs2_open@8158 387891d4-d844-
0410-90c0-e4c51a9137d3
```

```
commit 623b6ada2e723fbb20d38a75d9776660dc51850f
Author: Zacam <Zacam@387891d4-d844-0410-90c0-e4c51a9137d3>
Date:   Sun Jul 17 05:46:14 2011 +0000
    Update:
    libpng now at 1.5.2 (with update to pngutils)
    zlib now at 1.2.5
    (Files are in "LF" instead of "CRLF" this time around.)
    MSVC6-2010 and XCode project updates included.

    git-svn-id: svn://svn.icculus.org/fs2open/trunk/fs2_open@7352 387891d4-d844-
0410-90c0-e4c51a9137d3
```

```
commit 101018d8807fa4bbc83f7d63d55baf74a8319596
Author: Zacam <Zacam@387891d4-d844-0410-90c0-e4c51a9137d3>
Date:   Sat Feb 6 02:35:41 2010 +0000
    From Antipodes #5: PNG File Format support.
    Adds PNG to Maps (should only be used for Nameplates), Interface, Effects, H
UD (still needs HUD Rewrite to implement), Player/Squad Images, Multi Players, M
ulti Cache, CB Anims and Intel Anims.
    LIBPNG: Version 1.2.42, current and highest usable version (MAX_VERSION), Li
nux can use any 1.2.2x-1.2.42, but should NOT use 1.4.0 under any platform for A
NY reason (will require rewriting pngutils)
    ZLIB: Version 1.2.3.

    git-svn-id: svn://svn.icculus.org/fs2open/trunk/fs2_open@5890 387891d4-d844-
0410-90c0-e4c51a9137d3
```

図 2: fs2open 中で libpng を取り込んだことを示すコミットのログ

この基準に従うと，57 件のすべての場合においてファイルの対応を正しく検出できた．

57 件のすべてのファイルの組について，トークン単位の類似度と行単位の類似度はともに 1.0 であった．これらのファイルの間では，改行文字の変更以外に変更点は見つからなかった．

再利用元のリビジョンとして検出されたリビジョンは 3 件あり，そのリビジョンはすべてコミットメッセージと対応するリビジョンであった．

再利用元のリビジョンから，そのリビジョンに対応するコミットメッセージに記録されたリビジョンへの到達性について調べると，57 件中 44 件が到達性を有し，13 件が到達性を持たなかった．

fs2open でのコミットのチェックサムが 101018 であるものの中で到達性のない物は 18 件中 11 件あり，libpng 側でのコミットのチェックサムが 73176b であるものが 4 件，7836e2 であるものが 7 件であった．libng のリポジトリのコミットのコメントから，両者ともバージョンが 1.0 系のものであった．fs2open でのコミットのチェックサムが 101018 であるものは使用している libpng のバージョンが 1.2.42 であるとコメントに記述されていたが，libpng のリポジトリでは 1.0 と 1.2 が別のブランチであり，別々のブランチに同じファイルが存在し，コメントに記述されていたバージョンとは別のブランチのバージョンのファイルが検出されていたため，到達性を持たなかった．

fs2open でのコミットのチェックサムが 58f9e7 であるものの中で到達性のない物は 18 件中 2 件あった．該当するファイルは pngwtran.c および pngwutil.c であった．これらのファイルは，検出されたリビジョンとコメントに記述されたバージョンに対応するリビジョンでファイルの内容は同一であったが，それらのリビジョンの間でファイルの内容が編集された後，編集された箇所が元に戻されていたため，条件 2 を満たさなくなったものであった．

fs2open でのコミットのチェックサムが 623b6a であるものは，21 件すべての場合において到達性を有していた．

4.2 node および zlib

node(node.js) は Chrome の JavaScript 実行環境上のプラットフォームであり [12]，zlib は圧縮および解凍ライブラリのプロジェクトである [13]．

zlib のリポジトリについては，一部のコミットの日付が開発中の日付と異なっていると考えられたので，リポジトリ内の ChangeLog ファイルの内容からコミットの日付を修正した．

node のリポジトリの中に zlib を取り込んだコミットがあり，そのコミットのログを図 3 に示す．

コミットメッセージには zlib のバージョンまでは記されていなかった．ファイル deps/zlib/README.chromium の記述によれば，zlib のバージョンは 1.2.3 であった．

```
commit 5b8e1dabbc117b38c56f874f4e6d5b7e537cd3c1
Author: isaacs <i@izs.me>
Date:   Tue Sep 6 16:13:05 2011 -0700

    Initial pass at zlib bindings
```

図 3: zlib を取り込んだコミットの node リポジトリのコミットメッセージ

ztest28975.c

```
int hello() {return 0;}
```

再利用先として検出されたファイル

```
// Copyright (c) 2012 Google Inc. All rights reserved.
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.

int main() {
  return 0;
}
```

図 4: ztest28975.c および再利用先として検出されたファイル

手法により，延べ 65 件のファイルの起源情報を検出した．65 件のすべての場合において，再利用元のリポジトリを zlib，再利用先のリポジトリを node と検出した．

65 件のうち，再利用元のファイルを ztest28975.c と検出したものが 21 件あった．この 21 件について再利用元のリビジョンはすべて同一であった．一方，ztest28975.c の再利用先として検出されたファイルはパスやファイル名が様々であった．しかし，それらのファイルに対する ztest28975.c からの差分はコメントを除き同一であった．その一例を図 4 に示す．これら 21 件のファイルについてはファイルの内容が非常に短いため，再利用の関係にないファイル同士が類似度が設定した閾値以上になったと判断した．

44 件について，再利用先のファイルはすべて同一のリビジョンであった．

リポジトリのログから，zlib のファイルを deps/zlib/下にコピーしていることが分かった．このとき，再利用先のパスが再利用元のパスの先頭に deps/zlib/を加えたものであるのは 44 件中 35 件であり，9 件はこの条件を満たさなかった．

パスについて条件を満たした 35 件のうち，到達性があるのは 22 件であり，到達性のないものが 13 件あった．到達性がないものの原因のうち 1 件は，加えられた変更が後のコミットで戻されていたことである．残りの 12 件は，zlib ライブラリにおいてファイルで使用さ

```
    if (png_crc_finish(png_ptr, 0))
+   {
+      png_ptr->num_trans = 0;
       return;
+   }
```

図 5: 変更の部分的な取り込みが行われた前後の差分

```
 /* pngrutil.c - utilities to read a PNG file
  *
- * Last changed in libpng 1.2.17 May 12, 2007
+ * Last changed in libpng 1.2.17 May 14, 2007
```

```
 if (png_crc_finish(png_ptr, 0))
+   {
+      png_ptr->num_trans = 0;
       return;
+   }
```

図 6: 部分的に取り込まれたリビジョン間の差分

れる改行文字が変更されていたケースがであった．改行文字の差異は，類似度計算では無視されるため，手法では改行文字が変更される前のリビジョンを再利用元のリビジョンと検出していた．

4.3 変更の部分的な取り込みに関する調査

20 のリポジトリの組を調査し，変更の部分的な取り込みが行われているかを調査した．調査の結果，図 5 のような例が見つかった．行頭が+で始まる行が取り込まれた行である．一方，取り込まれた差分の元となったリビジョン間の差分は図 6 である．行頭が+もしくは-から始まる行が差分である．

この例では，再利用元のリビジョン間の差分 5 行のうち，3 行が取り込まれている．したがって，部分的な変更の割合は 0.6 である．

変更の部分的な取り込みの調査を行った 20 のリポジトリの組を対象に，部分的な変更の割合を調査した．値の高いものを調べてみると，変更の部分的な取り込みにより値が高く

なったのではなく，偶然同じ内容の変更が加えられていたため値が高くなってしまったことが多く見られた．例えば，以下のような場合である．

動作に影響を与えない修正 再利用元のリポジトリでソースコードのインデントの修正など，プログラムの動作に影響を与えないような修正が行われ，それとは関係なく再利用先のリポジトリでも同様の修正が行われた場合．

変更が削除のみの場合 再利用元リポジトリでリビジョン間の差分が行の削除のみの場合，再利用先リポジトリで独自の変更を加えるためにソースコードを編集した際に削除された行と一致してしまう場合がある．

したがって，本手法で求めた部分的な変更の割合の値が大きいからといって実際に変更が取り込まれているとはいえない場合が多いと考えられる．

5　まとめ

本研究では，与えられた 2 つのリポジトリに対して，あるファイルの再利用元リビジョンの情報である起源情報，起源情報に加えファイルに施された修正の情報である組成情報を検出する手法を提案した．ケーススタディの結果，1 つのケースに対しては，ファイルの対応および再利用の方向はすべて正しく，リビジョンの対応が正しく検出できたのは 8 割程度であった．もう 1 つのケースに対しても，半分程度のファイルについて起源を正しく特定できた．組成情報に関しては，変更の部分的な取り込みの値が意図せず高くなってしまう場合が多いという結果になった．

今後の課題として，組成情報の計算方法の改善を行うこと，再利用先のファイルの変更が再利用元のファイルへ取り込まれる場合の対応が挙げられる．

謝辞

本研究において，常に適切な御指導及び御助言を賜り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井上 克郎教授に心より深く感謝いたします.

本研究において，随時適切な御指導及び御助言を賜り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松下 誠准教授に心より深く感謝いたします.

本研究において，逐次適切な御指導及び御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石尾 隆助教に心より深く感謝いたします.

I would like to show my appreciation to Daniel Morales Germán, associate professor in the department of computer science at the University of Victoria.

本研究において，終始適切な御指導及び御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 KULA RAULA GAIKOVINA 特任助教に心より深く感謝いたします.

本研究において，様々な御指導及び御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神田 哲也氏に深く感謝いたします.

最後に，その他様々な御指導，御助言等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井上研究室の皆様に深く感謝いたします.

参考文献

[1] Ebart, C., Open Source Software in Industry, IEEE Software , vol.25, no.3, pp.52-53, 2008

[2] Godfrey M.W., Lijie Zou, Using origin analysis to detect merging and splitting of source code entities, IEEE Transactions on Software Engineering, vol.31, no.2, pp.166-181, 2005

[3] Inoue K., Sasaki Y., Pei Xia, Manabe Y., Where does this code come from and where does it go? - Integrated code history tracker for open source systems -, In Proceedings of the 34th International Conference on Software Engineering (ICSE) pp.331-341, 2012

[4] Pei Xia, Matsushita M., Yoshida N., Inoue K. Studying Reuse of Out-dated Third-party Code in Open Source Projects. コンピュータソフトウェア vol.30, no.4 (2013): pp.98-104.

[5] Ruffin, C. Ebert, C., Using open source software in product development: a primer, IEEE Software , vol.21, no.1, pp.82-86, 2004

[6] Kanda T., Ishio T., Inoue K: Extraction of Product Evolution Tree from Source Code of Product Variants, Proceedings of the 17th International Software Product Line Conference (SPLC2013), pp.141-150, 2013

[7] The Open Source Initiative — Open Source Initiative, http://opensource.org/

[8] FreeSpace Source Code Project - FS2Open, http://scp.indiegames.us/bnr_fs2open.php

[9] Git, http://git-scm.com/

[10] Japan Computer Emergency Response Team Cordination Center. https://www.jpcert.or.jp/

[11] libpng Home Page, http://www.libpng.org/pub/png/libpng.html

[12] node.js, http://nodejs.org/

[13] zlib Home Site, http://www.zlib.net/

[14] 佐々木 裕介, 山本 哲男, 早瀬 康裕, 井上 克郎: 大規模ソフトウェアシステムを対象としたファイルクローンの検出, 電子情報通信学会論文誌 D　Vol. J94-D No. 8 pp.1423-1433, 2011 年